Introduction to Motorola 68000's Addressing Modes

Daniele Paolo Scarpazza
daniele.scarpazza@elet.polimi.it

Politecnico di Milano
Last update: May 11th, 2005

Bibliography

- Textbook:

  Hamacher, Vranesic & Zaky
  Computer Organization
  McGraw-Hill Science
  August 2, 2001

- Reference manual:

  MOTOROLA M68000 FAMILY
  Programmer's Reference Manual
  © Motorola Inc., 1992

  Available for download at:

  `http://www.scarpaz.com/processors/`

- Acknowledgements

  Graphics and sample code adapted from:
  `http://goforit.unk.edu/asm/mc68000.htm`

Tools

- Tool:

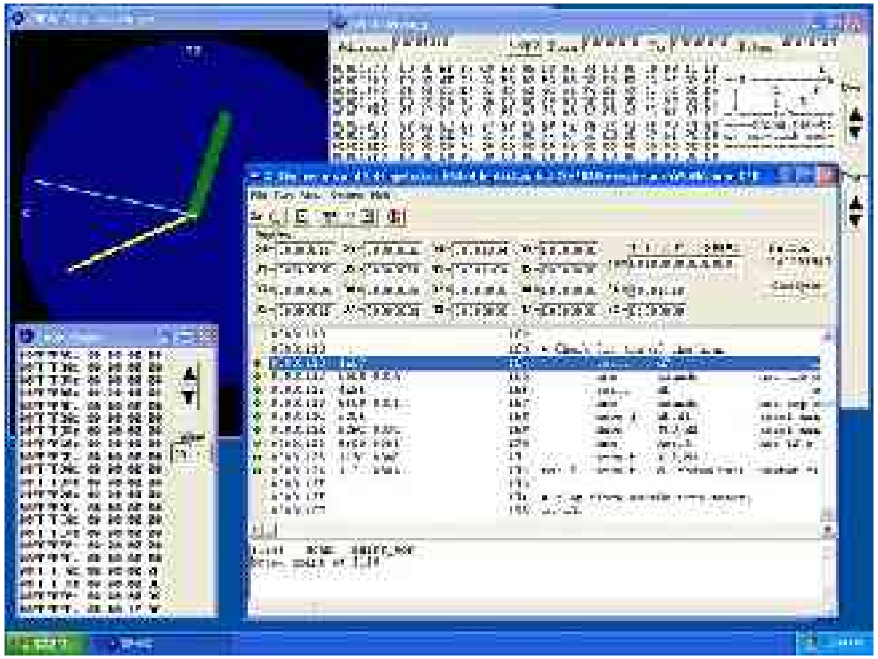
    EASy68K

    Editor/Assembler/Simulator for the 68000

  Available at:

    `http://www.monroeccc.edu/ckelly/easy68k.htm`

- The examples we provide here were successfully tested with this simulator (unless otherwise specified);



Motorola 68000 Assembly basics

- 8 data registers (D0-D7) and 8 address registers (A0-A7)
- The MOVE instruction has syntax:

  MOVE source, destination
- The stack in the 68000 family grows from higher to lower addresses;

  push = SP--;   pop= SP++;
- Address register A7 is the stack pointer.
- Function calls:
  - A6 is used as frame pointer;
  - D0 is used to return values to the caller;

Addressing modes

- Each instruction comprises an *operation code,* which specifies the function to perform;
- Instructions must also define *which are the operands* for that function;
- An instruction's *addressing mode* specifies the operands in one of the following ways:
  - by specifying *the value* of the operand;
  - by specifying *a register* that contains the operand;
  - by specifying how to derive the *effective address* of an operand in memory;
- Each addressing has its assembly language syntax;

Addressing modes: summary

- Register Direct
  - Data #1
  - Address #2
- Register Indirect
  - Address #3
  - Address with Postincrement #4
  - Address with Predecrement #5
  - Address with Displacement #6
- Address Register Indirect with Index
  - 8-Bit displacement #7
  - Base displacement #8
- Memory indirect
  - Postindexed #9
  - Preindexed #10
- Program Counter Indirect
  - with Displacement #11
- Program Counter Indirect with Index
  - 8-Bit displacement #12
  - Base displacement #13
- Program Counter Memory Indirect
  - Postindexed #14
  - Preindexed #15
- Absolute Data Addressing
  - Short #16
  - Long #17
- Immediate #18

Addressing modes

- Register <u>Direct</u> mode
  - #1: Data register direct mode
  - #2: Address register direct mode
- In the register direct modes, the instruction specifies the data or address register containing the operand;
- Assembly language syntax: **`Dn`** or **`An`**

Addressing modes

- #3: Address register <u>indirect</u> mode
  - the operand is in memory;
  - the instruction specifies which address register contains the address of the operand in memory;
- Assembly language syntax: **`(An)`**

Addressing modes

- #4: Address Register Indirect with Postincrement mode
  - the operand is in memory;
  - the instruction specifies which address register contains the address of the operand in memory;
  - after the operand address is used, it is incremented by 1, 2 or 4 depending on the operand size (byte, word, long word respectively)
  - if the address register is *stack pointer* and operand size is byte, the address is incremented by 2 to preserve alignment;
- Assembly language syntax: **`(An)+`**

Addressing modes

- #5: Address Register Indirect with Predecrement mode
  - the operand is in memory;
  - the instruction specifies which address register contains the address of the operand in memory;
  - before the operand address is used, it is decremented by 1, 2 or 4 depending on the operand size (byte, word, long word respectively)
  - if the address register is *stack pointer* and operand size is byte, the address is decremented by 2 to preserve alignment;
- Assembly language syntax: **`-(An)`**

Addressing modes

- #6 Address Register <u>Indirect with Displacement</u> mode
  - the operand is in memory;
  - the operand's address in memory is the sum of:
    - an address contained in an address register (the instruction specifies which register); and
    - a 16-bit displacement integer (the instruction specifies it)
- Assembly language syntax: **`(d, An)`**

Addressing modes

- Address Register Indirect with Index mode
  - #7 8-Bit Displacement
  - #8 Base Displacement
- The operand's address in memory is the sum of:
  - an address contained in an address register (the instruction specifies which register); and
  - a scaled index register (the instruction specifies which register); and
  - a 8-bit displacement or a base displacement integer (the instruction specifies it)
- Assembly language syntax: **`(d, An, Xn.s)`** where **`s`** is one of: **`B,W, L`**

Addressing modes

- #9: Memory Indirect Post-indexed mode
  - the operand is in memory and
    the operand's address is in memory too;
  - an intermediate address IA is obtained as:
    IA = address (in reg.) + base displacement (in instr.)
  - the operand is at the final address, obtained as:
    value @IA + index (in reg.) + outer displacement (in instr.)
- Assembly language syntax: **`([bd+An],Xn.s,od)`**
  where **`s`** is one of: **`B,W,L`**
  - all four user-specified values are optional;
  - if not specified, their value is assumed zero;

Addressing modes

- #10: <u>Memory Indirect Pre-indexed</u> mode
  - the operand is in memory and
    the operand's address is in memory too;
  - an intermediate address IA is obtained as:
    IA = address (in reg.) + base displacement (in instr.) +
    index (in reg.)
  - the operand is at the final address, obtained as:
    value @IA + outer displacement (in instr.)
- Assembly language syntax: **`([bd,An,Xn.s],od)`**
  where **`s`** is one of: **`B,W,L`**
  - all four user-specified values are optional;
  - if not specified, their value is assumed zero;

Addressing modes

- #11: Program Counter Indirect with Displacement mode
  - the operand is in memory;
  - the operand's address is the sum of the address in PC and a 16-bit displacement (in the instruction);
  - the operand is at the final address, obtained as: value @IA + outer displacement (in instr.);
  - this mode is allowed only for reads;
- Assembly language syntax: **`(d,PC)`**

Addressing modes

- <u>Program Counter Indirect with Index</u> modes
  - #12/#13: PC Indirect with Index (8-Bit/Base Displacement) are like modes #7/#8 Reg. Indirect with Index, except the PC is the base register;
  - the operand's address is the sum of the address in PC, an 8-bit or base displacement (in the instruction) and the scaled index (in the index register);
- Assembly language syntax: **`(d,PC,Xn.s)`** where **`s`** is one of: **`B,W,L`**

Addressing modes

- <u>Program Counter Memory Indirect</u> modes
  - #14/#15: PC Mem. Indirect Post-/Pre-index modes are like modes #9/#10 Memory Indirect Post-/Pre-index, except the PC is the base register;
  - the operand's address is the sum of the address in PC, an 8-bit or base displacement (in the instruction) and the scaled index (in the index register);
- Assembly language syntax: **`([bd,PC],Xn.s,od)`**
  `([bd,PC,Xn.s],od)`

  where **`s`** is one of: **`B,W,L`**

Addressing modes

- Absolute addressing modes
  - #16: Absolute Short Addressing mode;
  - #17: Absolute Long Addressing mode;
  - the operand is in memory;
  - the operand's address is a 16-/32-bit value in the instruction;
- Assembly language syntax:

```
(xxx).W
(xxx).L
```

Addressing modes

- #18: <u>Immediate</u> data;
  - the operand is in the instruction;
- Assembly language syntax: **`#xxx`**

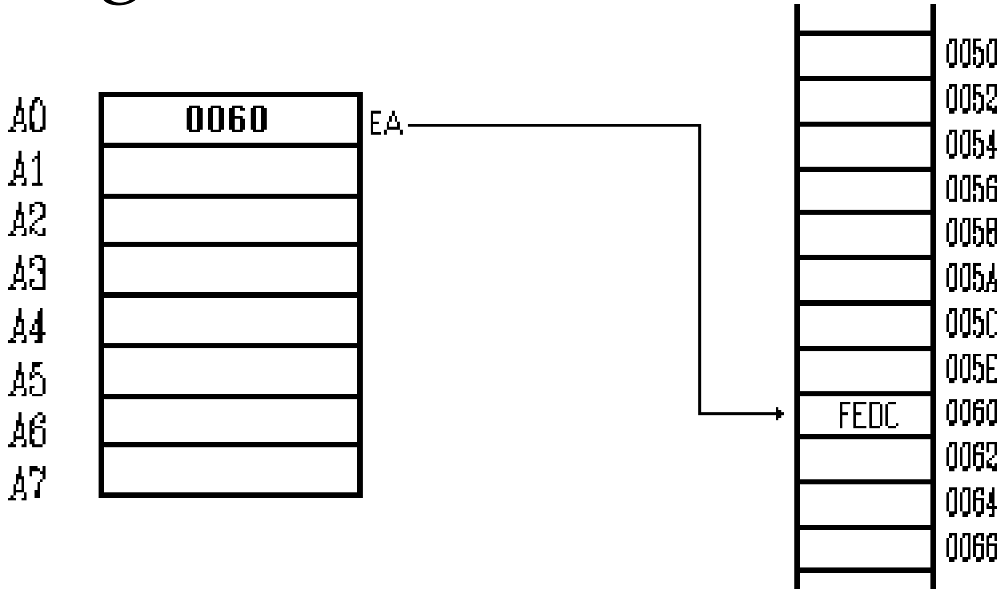
Addressing Mode examples

- Sample code: immediate and
  direct addressing modes

```
*** Example: ref000.X68

START   ORG     $1000

        CLR     D0              * clear value in D0 (0 --> D0)
        MOVE.W  #$7F0,D0        * move immediate word into data register
                                * EA of destination is data register direct
        MOVE.W  #$0008,A0       * move immediate word into address register 0
        ADDQ.W  #$0008,A1       * add immediate word into address register 1
        ADD.W   D0,A1           * add D0 to current contents A1
                                * EA of source is data register direct
                                * EA of destination is address register direct

        STOP    #$2000
        END     START
```

Addressing Mode examples

- Sample code: address register indirect mode

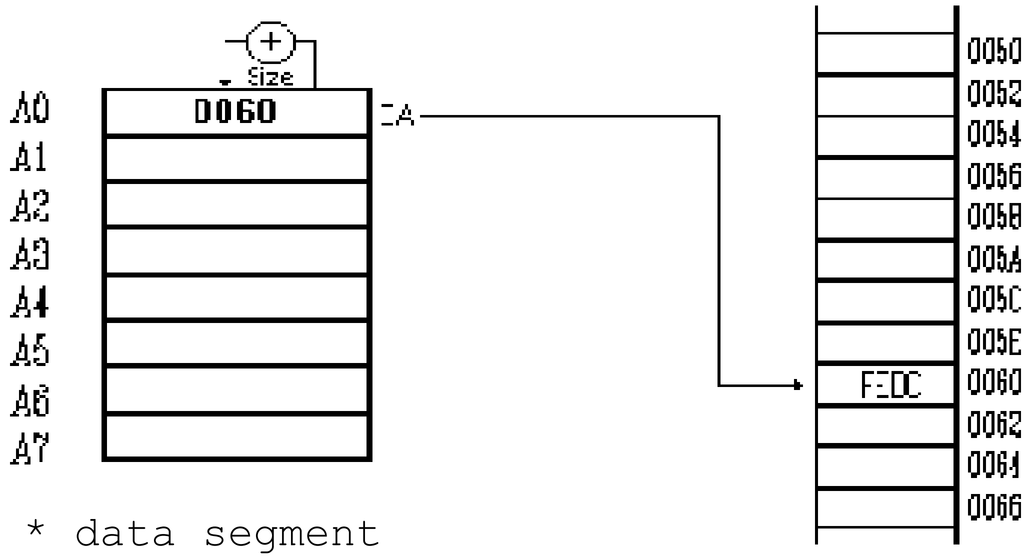


```
*** Example: ref001.X68
        ORG         $60             * data segment
DSEG    EQU         $60
        DC.W        $FEDC           * data word "FEDC"

START   ORG         $1000
        MOVE.W      #DSEG,A0        * point A0 to location $0060
        MOVE.W      (A0),D0         * load D0 from (A0), eg $0060

        STOP        #$2000
        END         START
```

Addressing Mode examples

- Sample code: address register indirect with postincrement mode

Useful to: scan tables
pop stack
`MOVE (A7)+, ...`



```
*** Example: ref002.X68
        ORG         $60                 * data segment
DSEG    EQU         $60
        DC.W        $FEDC               * load two words in subsequent locations
        DC.W        $BA98               *

START   ORG         $1000
        MOVE.W      #DSEG,A0            * point A0 to location $0060
        MOVE.W      (A0)+,D0            * load D0 from (A0), eg  $0060
        MOVE.W      (A0)+,D1            * load D1 from (A0), now $0062
                                        * now A0 is $0064
        STOP        #$2000
        END         START
```

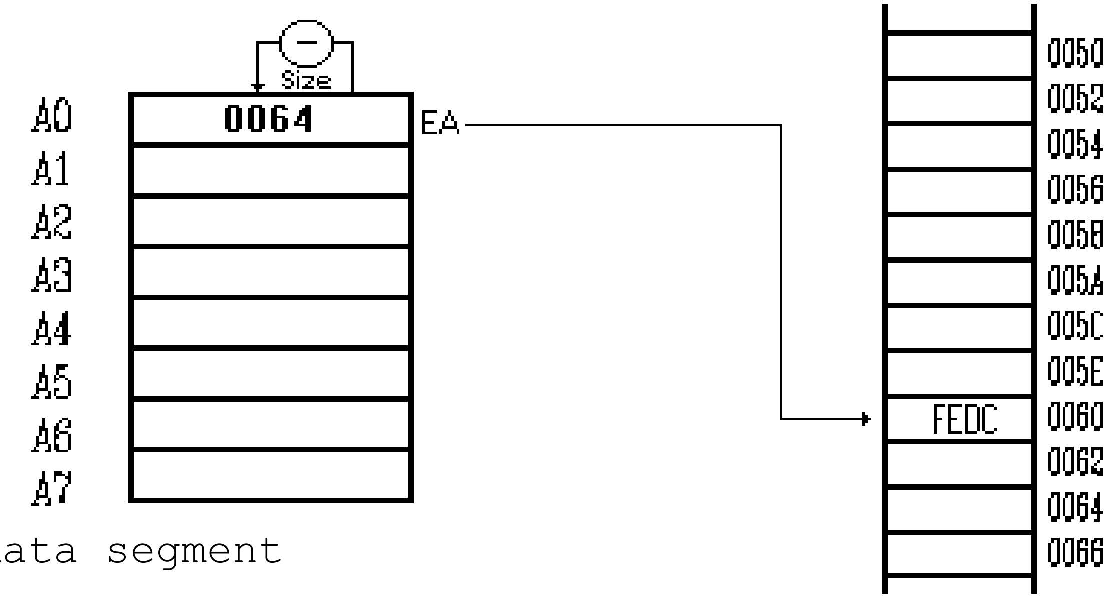
Addressing Mode examples

- Sample code: address register indirect with predecrement mode

Useful to: scan tables backward

push onto the stack

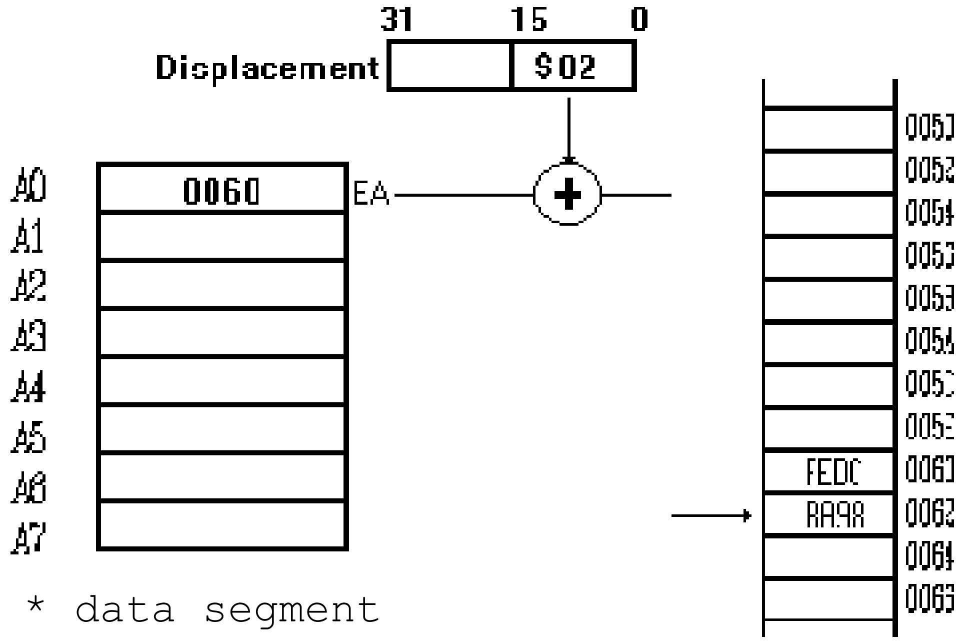
`MOVE ...,(A7)-`



```
*** Example: ref003.X68
        ORG $60                  * data segment
DSEG    EQU $60
        DC.W $FEDC               * load two words in subsequent locations
        DC.W $BA98               *

START   ORG     $1000
        MOVE.W  #DSEG+4, A0      * point A0 to location $0064
        MOVE.W  -(A0),D0         * A0 = A0-2 = $0062; load D0 from (A0)
        MOVE.W  -(A0),D1         * A0 = A0-2 = $0060; load D1 from (A0)

        STOP    #$2000
        END     START
```

Addressing Mode examples

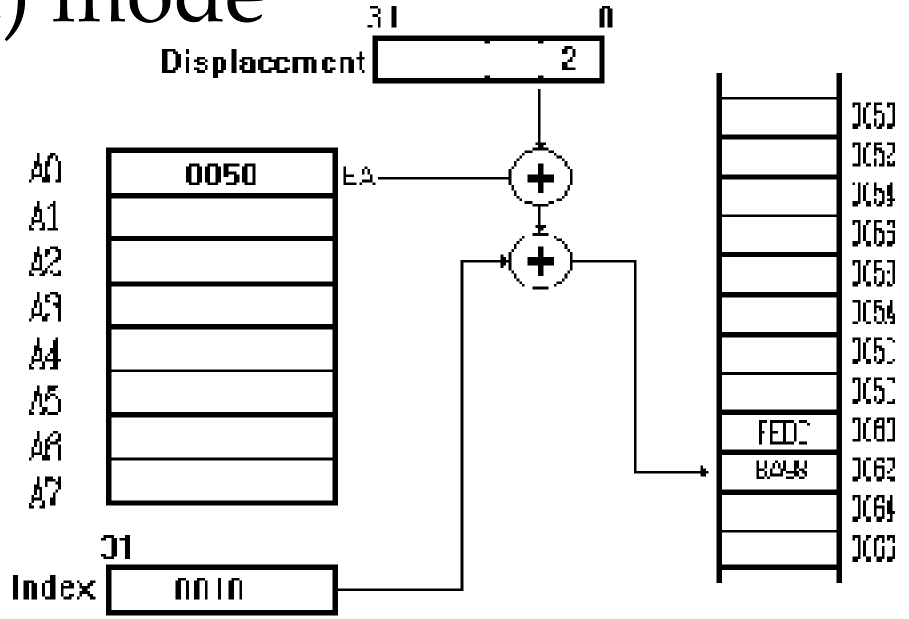
- Sample code: address register indirect with displacement mode



```
*** Example: ref004.X68
        ORG         $60              * data segment
DSEG    EQU         $60
        DC.W        $FEDC            * load two words in subsequent locations
        DC.W        $BA98            *

START   ORG         $1000
        MOVE.W      #DSEG, A0        * point A0 to location $0060
        MOVE.W      $2(A0),D0        * load the second word into D0

        STOP        #$2000
        END         START
```

Addressing Mode examples

- Sample code: address register indirect with index (8-bit) mode



```
*** Example: ref005.X68
        ORG         $60
DSEG    EQU         $60
        ORG         $70                * displaced data segment ($10 bytes later)
        DC.W        $FEDC              * load two words in subsequent locations
        DC.W        $BA98              *

START   ORG         $1000
        MOVE.W      #DSEG, A0            * point A0 to location $0060
        MOVE.W      #$10,  A1            * load A1 as index register
        MOVE.W      $02(A0,A1.W),D0  * indirect with index addressing mode

        STOP        #$2000
        END         START
```

Addressing Mode examples

- Sample code: absolute short addressing mode

```
*** Example: ref006.X68
        ORG        $60              * data segment
DSEG    EQU        $60
        DC.W       $FEDC            * load two words in subsequent locations
        DC.W       $BA98            *

START   ORG        $1000
        MOVE.W     #DSEG, A0        * point A0 to location $0060
        MOVE.W     DSEG+2,A1        * move (DSEG+2) to A1
        MOVE.W     A1,DSEG          * move A1 to (DSEG)

        STOP       #$2000
        END        START
```

Addressing Mode examples

- Sample code: absolute long addressing mode

```
*** Example: ref007.X68
*** Not designed to run in the simulator!!

        ORG       $60             * data segment
DSEG    EQU       $60
        DC.W      $FEDC           * load  two words in subsequent locations
        DC.W      $BA98           *

START   ORG       $1000
        MOVE      $7f000060,A0    * read from absolute location

        STOP      #$2000
        END       START
```

Addressing Mode examples

- Sample code: immediate addressing mode

```
*** Example: ref008.X68
*** Not designed to run in the simulator!!

CR              EQU $0A
LF              EQU $0D
PPI_INIT        EQU $7f03
PPI_CTRL_ADR    EQU $07ffffff                * address of the control register
PPI_DATA_ADR    EQU $07fffffe                * address of the control register

START     ORG       $1000
          MOVE.W    #PPI_INIT,D0             * move PPI init bytes to D0
          MOVE.L    #PPI_CTRL_ADR,A0         * move PPI control reg to A0
          MOVE.L    #PPI_DATA_ADR,A1         * move PPI data reg to A1
          MOVE.B    D0,PPI_CTRL_ADR          * initialise PPI
          ROR       #8,D0
          MOVE.B    D0,PPI_CTRL_ADR
          MOVE.B    #CR,D0
          MOVE.B    D0,(A1)                  * CR to PPI data reg

          STOP      #$2000
          END       START
```

Addressing Mode examples

- Sample code: program counter with displacement

Useful to access memory relative to the current value of the Program Counter.
Example: jumps in position independent code,
reading constants in code segments

```
*** Example: ref009.X68

START           ORG     $1000
                JMP     TABLE_END

    TABLE:      DC.B    $20                 * table inside the code segment
                DC.B    $32                 *
                DC.B    $64                 *
    TABLE_END:
                MOVE.B  TABLE,   D0         * moves TABLE[0] into D0
                MOVE.B  TABLE+1, D1         * moves TABLE[1] into D1

                STOP    #$2000
                END     START
```

Addressing Mode examples

- Sample code: program counter with index

This addressing mode extends the program counter relative mode to include an index and offset value.

```
*** Example: ref010.X68

START           ORG       $1000
                JMP       TABLE_END
     TABLE:     DC.B      $20
                DC.B      $32
                DC.B      $64
     TABLE_END:

                MOVE      #0,A0            * use A0 as index register
                MOVE.B    TABLE(A0),D0     * read TABLE[0] into D0
                ADD       #1,A0            * use A0 as index register
                MOVE.B    TABLE(A0),D1     * read TABLE[1] into D1
                STOP      #$2000
                END       START
```