Appendix A: MC 68000

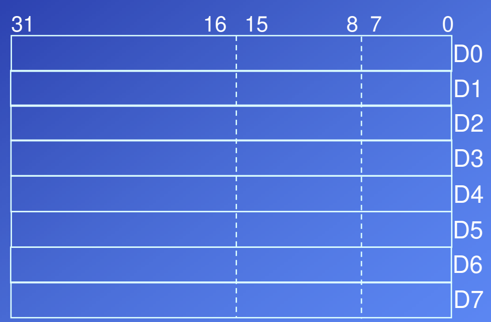
The 68K will be used to illustrate some topics discussed in class.

- 16-bit data bus (can operate in 8-bit mode when necessary)
- 16/32-bit microprocessor
  Internally 32-bit data paths and instructions, but interfaces with external components using a 16-bit data bus, so, a programmer considers it 32-bit chip while a system designer considers it a 16-bit chip)
- 16 32-bit registers (eight data and eight address registers)
- 24-bit address bus: These 24 lines can therefore address 16 MB of physical memory with byte resolution
- Operations can be performed on 5 different data types:
  - Bit, *byte*, 16-bits (*word*), 32 bits (*long word*), BCD
- Memory-mapped input/output (I/O)
- 14 addressing modes
- Two modes of operation: *Supervisor* vs. *User*
  - Some instructions cannot be executed in user mode
  - Access to memory can be restricted by connecting the FCO (functions code output) pins to the memory address decoding circuitry.

Programmable Registers (*User Programmer's Model*)

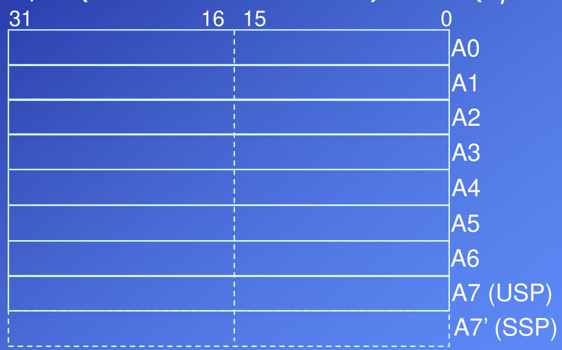
Data Registers:

- consist of 8 identical registers
- can be addressed as 8, 16, or 32 bits



Address Registers:

- 8+1 registers (A0 to A7 and A7'). These are typically used as pointers.
- The address registers can only be used as 16 or 32 bits.
- The A7 register is also the stack pointer. It is duplicated for the user and supervisor states, i.e, A7 (User Stack Pointer -USP) and A7' (System Stack Pointer -SSP).



Since the address bus is 24 bits wide, only the first 24 bits of the data in an address register is used.

When the low-order word (16 bits) in an address register is used, these bits are sign-extended to 24 bits before being placed on the address bus.

Status Register:

- 16 bits
- Consists of two parts: System and user (*CCR Condition Code Register*)

- Condition codes: Overflow (V), Zero (Z), Negative (N), Carry (C), Extend (X).
- Interrupt mask (I_0 I_1 I_2)
- Additional status bits indicating that the processor is in Trace (T) mode and/or in the Supervisor (S) state
- Bits 5, 6, 7, 11, 12, 14 are undefined and reserved for future expansion.

Program Counter (PC):

- 32 bits
- Can also be used as an address register

Data Organization in Memory

High-order parts of data are placed in memory starting from lower addresses.

Address	15 … 8		7 … 0	Long word
$000000	Byte 0	WORD 0	Byte 1	LONG WORD 0
$000002	Byte 2	WORD 1	Byte 3	LONG WORD 0 / LONG WORD 1
$000004	Byte 4	WORD 3	Byte 5	LONG WORD 1
$000006	Byte 6	WORD 4	Byte 7	

- Bytes are individually addressable.
- The high-order byte of a word has the same address as the word.
- The low-order byte has an odd address, one count higher.
- Instructions and multibyte data are accessed only on word (even byte) boundaries.
- Each *word* (16 bits) or *long word* (32 bits) must start at even address.
- If a long-word operand is located at address n (n even), then the second word of that operand is located at address n+2.

Addressing Modes

The 68000 supports 14 different addressing modes derived from six basic types:

1. Register Direct
2. Immediate
3. Absolute
4. Register Indirect
5. Program Counter Relative
6. Implied

1a. Data Register Direct

The operand is in a data register (whose name is given directly).

MOVE.W D_n , D_m $D_n \rightarrow D_m$

B: *Byte*, W: *Word*, L: *Long*

1b. Address Register Direct

The operand is in an address register (whose name is given directly).

If the destination is an address register, the instruction ends with an "A."

MOVEA.W D1 , A5 $D_1 \rightarrow A_5$ (Source data register, dest. addr. register)

The data may only be W: *Word* or L: *Long* .

2a. Immediate

The actual data to be used as the operand is included in the instruction itself.

MOVE.L #$4A7F0000 , D0 ; move the immediate data $4A7F0000 to D0

2b. Quick Immediate

Can only be used with some instructions.

The source operand must use immediate mode, and only with an 8-bit signed integer constant (-128, ..., 127). The destination must be a D register.

The instruction takes up less space (2 bytes, not 6) and works faster.

For example, it is used for the MOVE instruction on 8-bit data.

MOVEQ #5 , D0 ; 32 bits of D0 are affected by this instruction

3a. Absolute Short

The instruction provides the 16-bit address of the operand in memory. The 16-bit address is sign-extended to 24 bits.

MOVE.B D0, ($58AA) ; written to address $0058AA

MOVE.B D0, ($B51A) ; written to address $FFB51A

3b. Absolute Long

Used when the address size is more than 16 bits.

The instruction provides the 24-bit address of the operand in memory.

MOVE.W ($45C720),D7 ; 16 bits starting at location $45C720 written to D7

4. Register Indirect

4a. Address Register Indirect

An address register contains the address of the source or destination operand.

31 0

[A_n] → [data]

Example:

Before the execution of the instruction:

31............ 0

Registers	
D0	4350 A7C8
A0	0000 1000

Content of D0 written to the address A0 points to.

Memory	
00100	XX XX
00102	XX XX
00104	XX XX

The content of A0 does not change.

High-order parts of data are placed in memory starting from lower addresses.

After the instruction has been run, the state of memory:

MOVE.B D0, (A0)

Memory	
00100	C8 XX
00102	XX XX

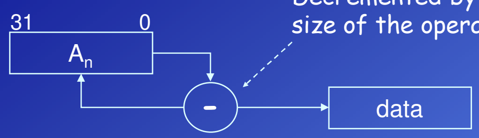
MOVE.W D0, (A0)

Memory	
00100	A7C8
00102	XX XX

MOVE.L D0, (A0)

Memory	
00100	4350
00102	A7C8

4b. Address Register Indirect with Predecrement



Decremented by 1, 2, or 4, based on the size of the operand.

$A_n - (1, 2, 4) \rightarrow A_n$
B W L

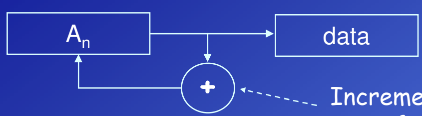
Example:

MOVE.W D0 , - (A0) ; First, A0 is decremented by 2, then the
; content of D0 is written to where A0 points to.

A0: 00001002, D0: 3725A100
After the instruction has been run, the state of memory:
001000: A1
001001: 00
After the instruction has been run, A0=00001000.

The predecrement mode can be used for array operations.
It can also be used for writing to the top of a stack (PUSH).

4c. Address Register Indirect with Postincrement



Incremented by 1, 2, or 4, based on the size of the operand.

$A_n + (1, 2, 4) \rightarrow A_n$
B W L

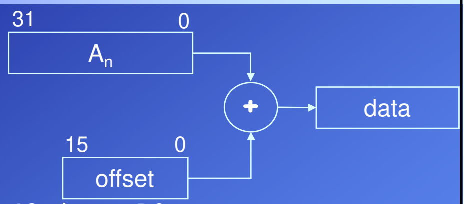
Example:

MOVE.W (A0)+ , D0 ; First, the 16-bit data A0 points to is written to D0,
; then, A0 is incremented by 2.

A0: 00001000, D0: XXXXXXXX
Memory:
001000: A1
001001: 00
After the instruction has run, A0=00001002, D0: XXXXA100.

Can be used for reading from the top of a stack (PULL).
Predecrement and postincrement modes are used for stack and queue operations. The 68000 does not provide special stack instructions.

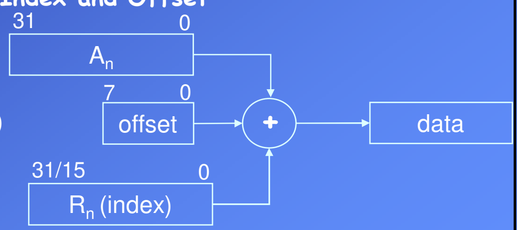
4d. Addr. Register Indirect with Offset



Offset: 16-bit signed number.
It is possible to address memory locations up to 32K bytes after (or 32K bytes before) the location A_n points to.
Examples:

```
MOVE.B   - 4(A2) , D0       ; <ea>= -4 plus A2, data to D0
MOVE.W   $0C(A5) , D7       ; <ea>=$0C+A5, data to D7
MOVE.L   -2(A3) , 12(A5)    ; <ea>= -2 plus A3, data to 12+A5
```

4e. Address Register Indirect with Index and Offset



Can be A_n or D_n.
16- or 32-bit parts can be used.

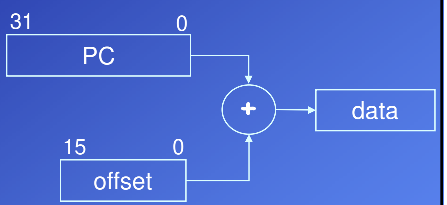
Example:

MOVE.W - 2(A3,D5.W) , 4(A5)

- A3: Base
- D5: Index
- -2 : Offset

- <ea>=-2+(A3)+(D5.W)
- Data to 4+(A5)

5a. Program Counter Relative with Offset



Fixed (absolute) addresses are not used.

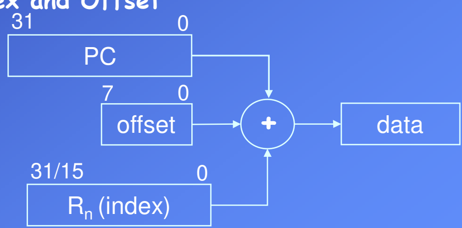
The address of data is determined relative to the address of the instruction being run.

The program can still run placed in different addresses.

Example:

```
MOVE.B   50(PC) , D5   ; 50 acts as an index, PC is the Program Counter
```

5b. Program Counter Relative with Index and Offset



Can be A_n or D_n.
16- or 32-bit parts can be used.

Example:
MOVE.W - 2(PC,D5.W) , 4(A5)
PC: Base
D5: Index
-2 : Offset

6. Implied Register

These instructions require no operands, although they may store or retrieve data from the stacks.
Examples: RTS, TRAPV, NOP

Instruction Format in MC68000

All instructions will be multiples of 16 bits.

Each instruction is at least 1 word, at most 5 words.

The instruction code (or op word) specifies the instruction's operation and addressing modes of the operands.

15 ... 0
Op word
Immediate data (if any, one or two words)
Source addr. extension
Dest. addr. extension

Instruction Format Examples:

Single Operand Instruction Format:

Instruction word:

15 … 8	7 6	5 4 3	2 1 0
Opcode	Size	Mode	Register

Size: 00: B, 01: W, 10: L

Refer to the reference manual for the codes in the Mode and Register fields.

Example:

CLR.W D3 — 01000010 01 000 011 (CLR W D 3)

Examples continued:

CLR.L (A2)+ — 01000010 10 011 010 (CLR L (An)+ 2) — Address register indirect postincrement

CLR.B ($3000) — 01000010 00 111 000 (CLR B Absolute short) — Absolute addressing (short)
0011 0000 0000 0000 — The address ($3000) is in the second word.

CLR.B $4(A6) — 01000010 00 101 110 (CLR B d(An) 6) — Address register indirect with offset
0000 0000 0000 0100 — The offset ($4) is in the second word as 16 bits.

CLR.B -7(A6) — 01000010 00 101 110 (CLR B d(An) 6) — Address register indirect with offset (negative offset)
1111 1111 1111 1001 — The offset (-7) is in the second word as 16 bits.

Additional Word for Register Indirect with Index and Offset Addressing Mode:

In the register indirect with index and offset addressing mode, there is an additional word.

The additional word contains information about the index register and offset.

15	14 13 12	11	10 9 8	7 … 0
D/A	Index reg.	W/L	0 0 0	offset
0:D 1:A		0:W 1:L		8 bits reserved for offset

Example:

CLR.W $C2(A3, D7.L)

01000010 01 110 011
0 111 1 000 11000010

D 7 L $C2

CLR.W d(A3,Rn.S)

The additional word contains information about the index register and offset (displacement).

Instruction Format Examples (continued)

Two-Operand Instruction Format:

Example: MOVE

15 14	13 12	11 10 9	8 7 6	5 4 3	2 1 0
0 0	Size	Reg.	Mod.	Mod.	Reg
	01: B 11: W 10: L	source	source	destination	destination

The instruction is not decoded based on just this field; the whole opword is used.

Example:

MOVE.W D2, (A5)+

00 11 101 011 000 010

W 5 (An)+ D 2

Example:

MOVE.W #$1234 , $25(A3)

W 3 d(An) immediate

00 11 011 101 111 100	
0001 0010 0011 0100	$1234 (immediate data)
0000 0000 0010 0101	$0025 (source address extension)

Quick Instruction Format:

MOVEQ

0111	Reg.	0	data

Bits: 15, 11 10 9, 8, 7, 0

Example:
MOVEQ #-5 , D2

0111	010	0	1111 1011

-5

- 8-bit immediate data used.
- Source is a data register.
- 32 bits of the register are affected by this instruction.
- Takes up less space than normal immediate addressing.

The normal MOVE instruction that performs the same operation takes up 3 words (48 bits) of space.

MOVE.L #-5 , D2

L 2 D immediate

00 10 010 000 111 100
1111 1111 1111 1111
1111 1111 1111 1011

32-bit "-5"

ADDQ operates on 3-bit immediate data.

MC68000 Instructions

In this section, we will introduce some MC68000 instructions.

Data Movement Instructions:

MOVEM ***Move multiple registers***

- Writes all specified registers to memory starting at a specific address, or reads data from specified memory address and places them in specified registers.
  Syntax 1: MOVEM <register list>,<ea>
  Syntax 2: MOVEM <ea>,<register list>
  Examples:
```
MOVEM.L  D0-D7/A0-A6 , $1234   ; save D0-D7/A0-A6 to
                               ; memory starting at $1234
MOVEM.L  (A5) , D0/D5/A0-A3    ; read D0, D5, A0-A3,
                               ; from memory address pointed by A5
```

Can be used to save working registers on entry to a subroutine and to restore them at the end of a subroutine.

```
MOVEM.L D0-D5/A0-A3,-(A7)     ; Push registers D0-D5/A0-A3 onto the stack
    ...
    Body of subroutine
    ...
MOVEM.L (A7)+,D0-D5/A0-A3     ; Restore registers D0-D5/A0-A3 from the stack
RTS                           ; Return to the calling program
```

LEA Load effective address

Operation: [An] ← <ea>

Used to copy the address of a variable into an address register.

All 32 bits of the address register are affected by this instruction.

Sample syntax:

```
LEA Table,A0            ; register A0 will point to the
                        ; beginning of Table
LEA (Table,PC),A0       ; calculates effective address of
                        ; Table w.r.t. to PC, deposits it in A0.
LEA (-6,A0,D0.L),A6     ; calculates A0+D0.L sign-extended
                        ; to 32 bits minus 6, deposits it in A6.
LEA (Table,PC,D0),A6
```

Example:

```
        LEA    ARRAY , A0      ; Array address to A0
        MOVE.B (A0)+, D1       ; Load first element of array to D1,
        ...                    ; increment A0 to point to next elmt.
ARRAY   DS.B   100             ; Define Storage (directive)
```


Flow Control Instructions:

Bcc Branch on condition cc

cc specifies the condition.

If cc = 1 THEN [PC] ← [PC] + d

d: 8- or 16-bit signed offset.

Reminder: When the instruction is being run, PC points to the instruction after Bcc.

Syntax: Bcc <label>

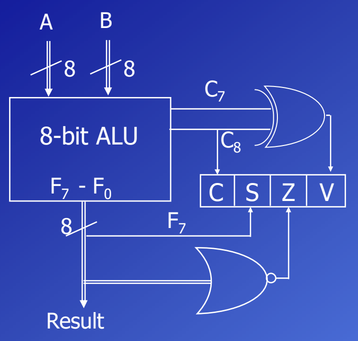
Relative size can be specified if needed: BEQ.B (EQual) or BNE.W (Not Equal)

If the size is not specified, the compiler computes the relative address of an appropriate size based on the distance of the label.

Conditions (cc):

BCC	branch on carry clear	branch if $C = 0$
BEQ	branch on equal	branch if $Z=1$
BGT	branch on greater than	branch if $(Z + (N \oplus V)) = 0$
BHI	branch on higher than	branch if $(C + Z) = 0$
BGE	branch on greater than or equal	branch if $(N \oplus V) = 0$
BLT	branch on less than	branch if $(N \oplus V) = 1$
BLS	branch on lower than or same	branch if $(C + Z) = 1$

Setting of Flags:



Overflow:
V= $C_7 \oplus C_8$
C_8: Carry
C_7: Carry in the previous bit

Overflow can also be determined based on:

pos + pos → neg pos – neg → neg
neg + neg → pos neg – pos → pos

In subtraction and comparison operations, the carry **C** bit serves as the BORROW flag.

Reminder:

Carry: May result from the addition of unsigned numbers. Indicates that the result does not fit into n bits and an (n+1)st bit is needed.

Borrow: May result from the subtraction of unsigned numbers. Indicates that the first number is smaller than the second.

In subtraction using two's complement, if an (n+1)st bit forms, there is no borrow.

Overflow: May form only in the addition or subtraction of signed numbers. Indicates that the result cannot be expressed using the allotted number of bits.

DBcc Test condition, decrement, and branch

Syntax: DBcc Dn,<label>

Here, the label is a 16-bit relative address.

16 bits of Dn is used as a counter.

Operation:

IF(condition cc false)
  THEN [Dn] ← [Dn] - 1 (*decrement loop counter*)
    IF [Dn] = -1 THEN instruction after DBcc (PC incremented by 2 in fetch cyc.)
          ELSE [PC] ← [PC] + d (branch relative)
  ELSE instruction after DBcc (PC incremented by 2 in fetch cycle.)

Example: Loop (10 times)

```
      MOVEQ  #9 , D0     ; Start value 9, because exiting on D0=-1
L1    ......             ; Inside the loop
      ......
      DBF    D0,L1       ; Here, F: False, condition always false,
                         ; branches if false
```

Example: Comparing Two Arrays (Are all elements equal?)

The first array starts at address ARRAY1, the second starts at address ARRAY2.
The arrays have 50 8-bit elements.
The contents of the arrays have been filled in before the program starts.

DBcc exits on -1

```
        LEA       ARRAY1, A0      ; Start addresses of the arrays
        LEA       ARRAY2, A1      ; A0 points to ARRAY1, A1 points to ARRAY2
        MOVE.W    SIZE, D0        ; Size of arrays
        SUBQ.W    #1, D0          ; Decrement D0 by 1 for use in DBNE later
LOOP    CMPM.B    (A0)+, (A1)+    ; Array elements compared as pair of bytes
        DBNE      D0, LOOP        ; Test, decrement D0, and loop until not equal
        TST.W     D0              ; Why did loop exit? (D0?), sets N &Z based on D0
        BMI       EQUAL           ; Branch if neg. (If D0=-1 on exit, all elmts. equal)
DIFFERENT .......
        .......
EQUAL   .......
        .......
ARRAY1  DS.B      50              ; Allocate memory for elements of 1st array: 50B
ARRAY2  DS.B      50              ; Allocate memory for elements of 2nd array: 50B
SIZE    DC.W      50              ; Define constant in memory of length one word
                                  ; 50 elements in each array
```