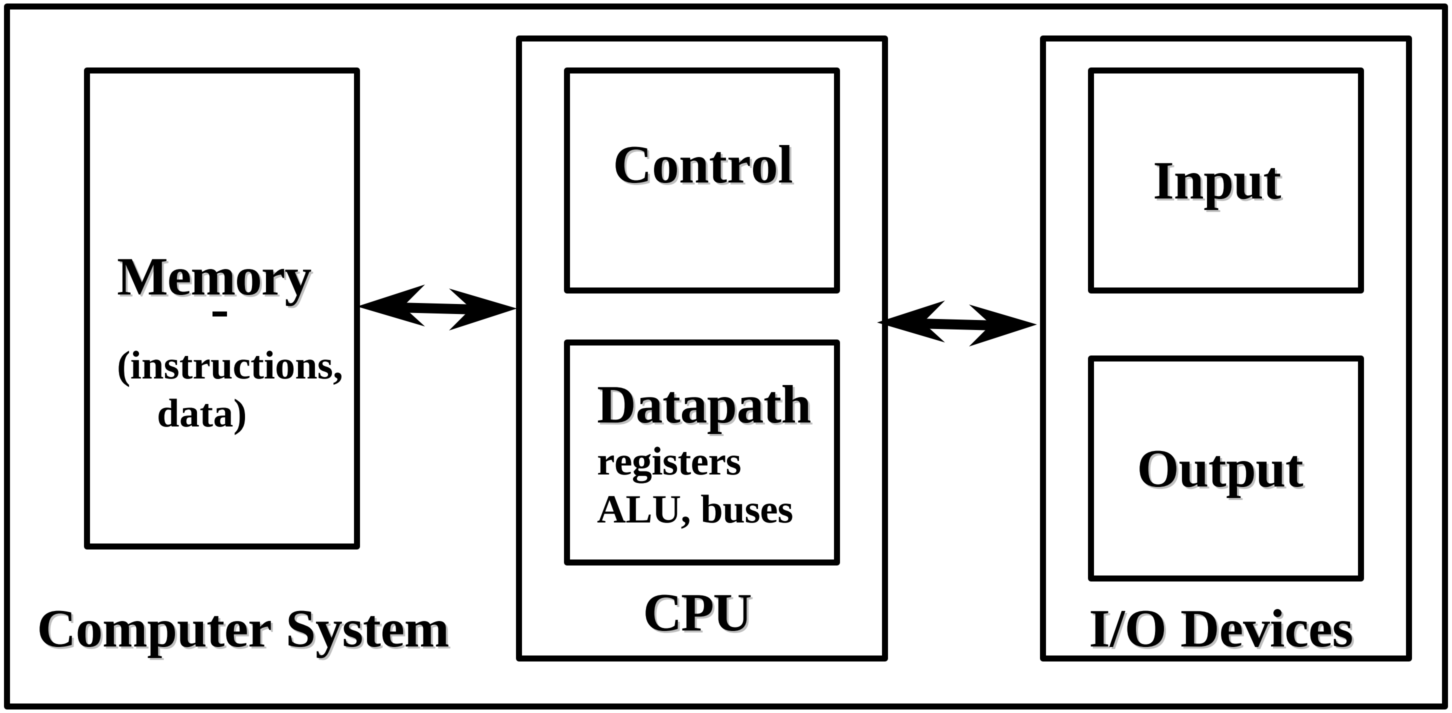
Basic computer operation and organization

- A computer manipulates binary coded data and responds to events occurring in the external world (users, other devices, network). This is called a stored-program, or a Von-Neumann machine architecture:
  - Memory is used to store both program instructions and data (this is the core of the Von-Neumann architecture).
  - Program instructions are binary coded data which tell the computer to do something, i.e. add two numbers together.
  - Data is simply information to be used by the program, i.e. two numbers to be added together.
  - A central processing unit (CPU) with the following tasks:
    - Fetching instruction(s) and/or data from memory
    - Decoding the instruction(s)
    - Performing the indicated sequence of operations

The Von-Neumann Computer Model

- Partitioning of the computing engine into components:
  - Central Processing Unit (CPU): Control Unit (instruction decode , sequencing of operations), Datapath (registers, arithmetic and logic unit, buses).
  - Memory: Instruction and operand storage
  - Input/Output (I/O)
  - The stored program concept: Instructions from an instruction set are fetched from a common memory and executed one at a time

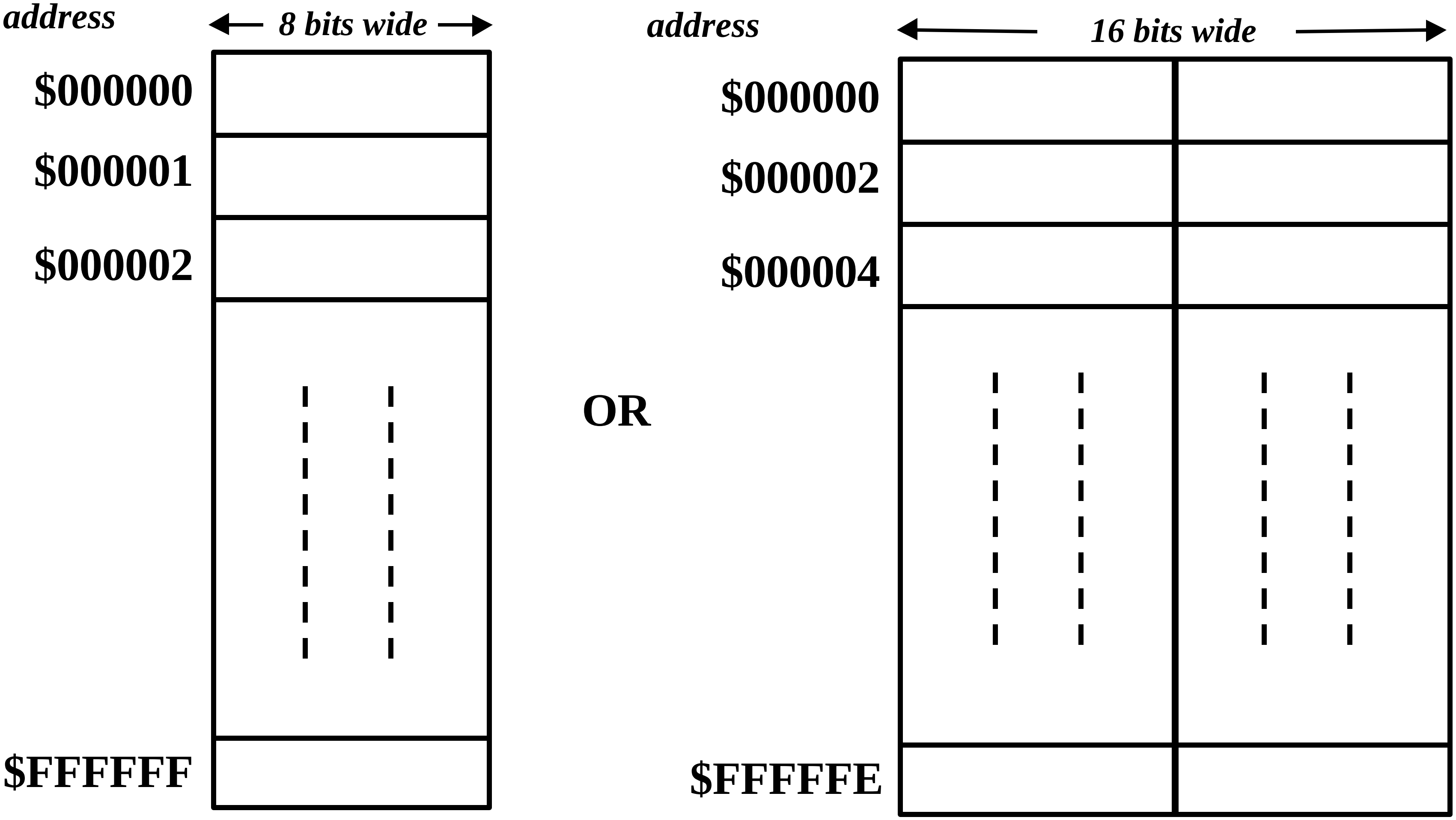


Central Processing Unit (CPU)

Control Unit	Arithmetic Logic Unit (ALU)	Registers

- **Control unit**
  - Decodes the program instructions.
  - Has a program counter which contains the location of the next instruction to be executed.
  - Has a status register which monitors the execution of instructions and keeps track of overflows, carries, borrows, etc.
- **Arithmetic Logic Unit**
  - Carries out the logic and arithmetic operations as required for instructions decoded by the control unit.
- **Registers:**
  - Program counter, status registers, stack pointer for subroutine use.
  - A number of general-purpose registers accessed by instructions to store addresses, instruction operands, and ALU results

Vertical Grid Organization of Memory



address ← *8 bits wide* →

$000000

$000001

$000002

$FFFFFF

OR

address ← *16 bits wide* →

$000000

$000002

$000004

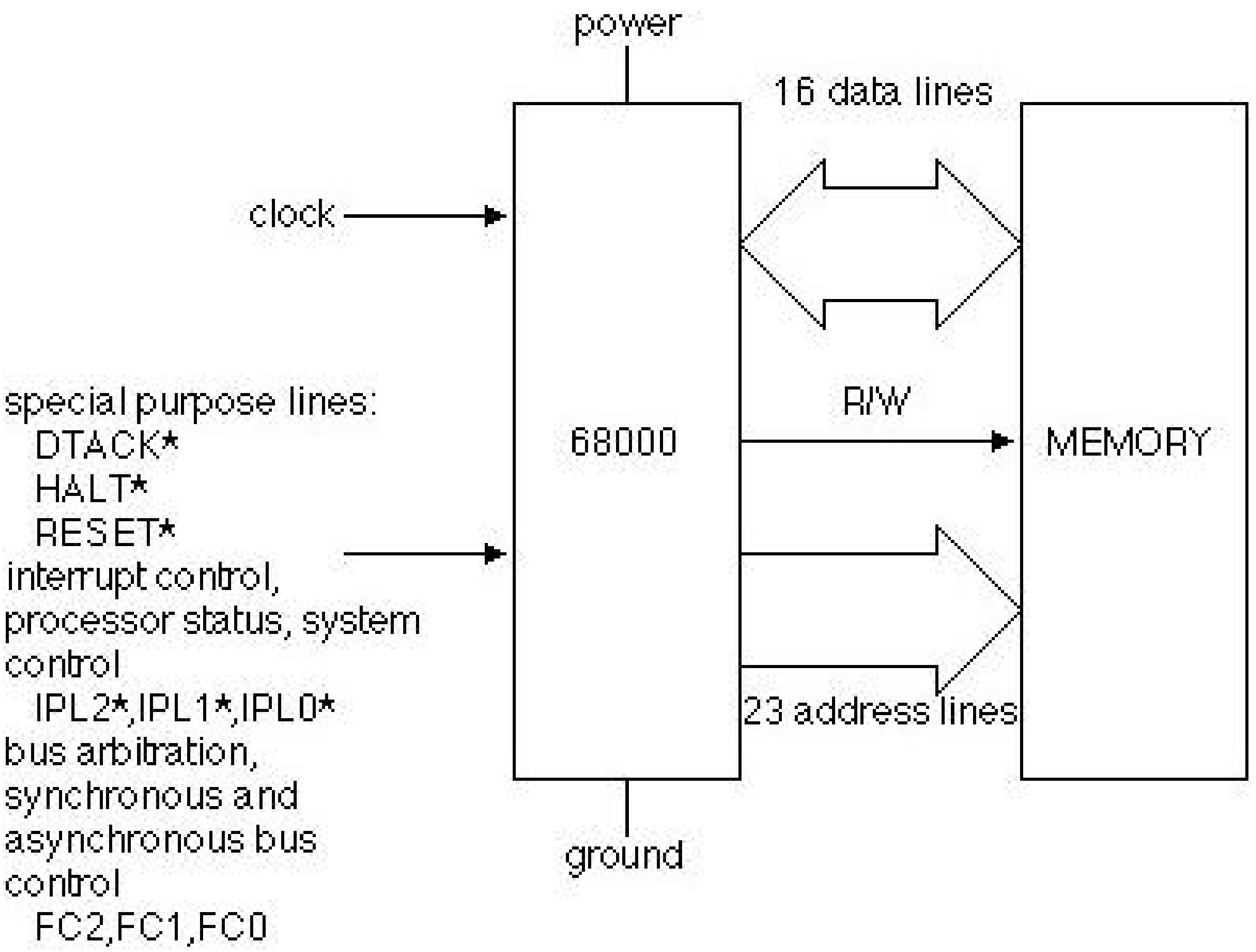
$FFFFFE

The memory grids above represent
\$FFFFFF = 2^{24} = 16,777,216 bytes of memory

Computer Data Storage Units

- **Bit** - Smallest quantity of information that can be manipulated inside a computer; value is either 0 or 1.
- **Byte** - Defined to be a group of 8 bits; typically the minimum size required to store a character.
- **Word** - Basic unit of information stored in memory and processed by a computer. Typical computer word lengths are 16, 32 and 64 bits.
- For the 68000, a word is 16-bits , and a long word is 32 bits.
- Words and long words in the 68000 must start at *even memory addresses* (e.g. $1000 is allowed, but $1001 produces a memory alignment error).

68000 Architecture



power

16 data lines

clock

68000

R/W

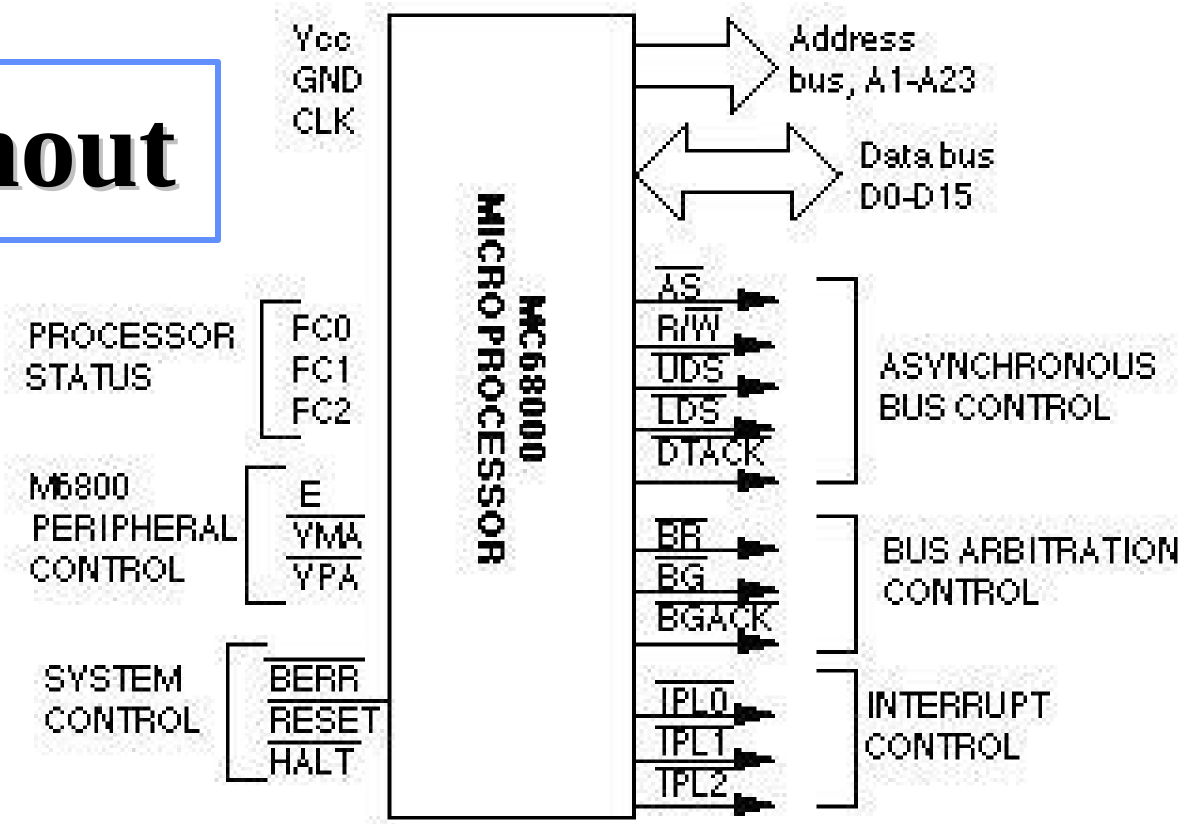
MEMORY

23 address lines

ground

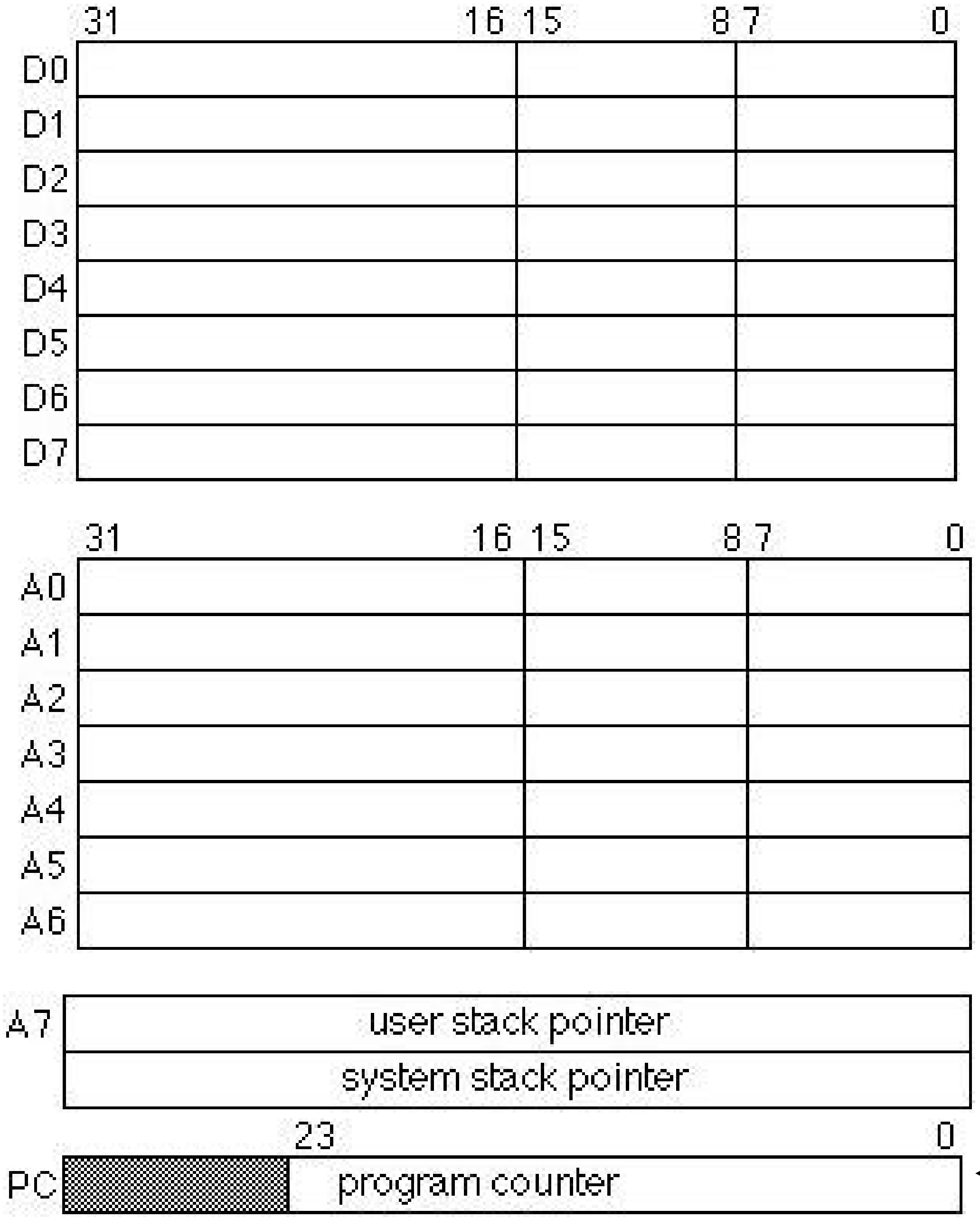
special purpose lines:
DTACK*
HALT*
RESET*
interrupt control,
processor status, system
control
IPL2*,IPL1*,IPL0*
bus arbitration,
synchronous and
asynchronous bus
control
FC2,FC1,FC0

68000 Pinout



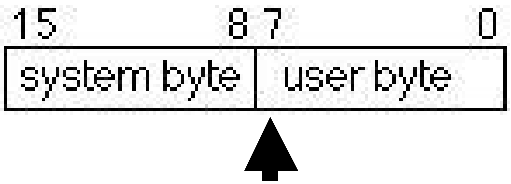
Pin Name	Descriptions
D0-D15	Data Bus
A1-A23	Address Bus
$\overline{\text{AS}}$	Address Strobe(Indicates value on address bus is valid)
R/ $\overline{\text{W}}$	Read/Write control
$\overline{\text{UDS}}$, $\overline{\text{LDS}}$	Upper byte, Lower byte Data Strobes
$\overline{\text{DTACK}}$	Data Transfer Acknowledge
FC0-FC2	Function Code (status) options
CLK	System Clock

68000 Internal Register Organization



Data registers

Address registers



Status register

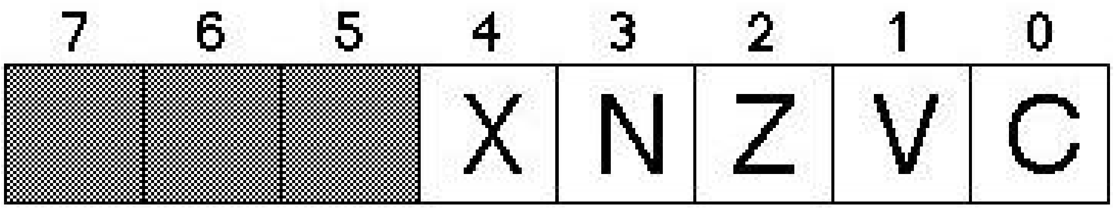
Stack pointers

Program counter

Status Register: Condition Code Register (CCR)


16-bit status register

15 — 8	7 — 0
Systems information	CCR



7	6	5	4	3	2	1	0
▨	▨	▨	X	N	Z	V	C

bits	function
7,6,5	not used
4	extend bit retains carry bit for multi-word arithmetic
3	negative set to 1 if instruction result is negative, set to 0 if positive
2	zero set to 1 if result is 0
1	overflow set if signed overflow occurs
0	carry/borrow

Status Register: The System Part

15	14	13	12	11	10	9	8
T	▨	S	▨	▨	I_2	I_1	I_0

bits	function
11,12,14	not used
8,9,10	interrupt mask a priority scheme to determine who has control of the computer
13	supervisor set to 0 if user, set to 1 if supervisor
15	trace set to 1 if program is to be single stepped

Description of 68000 Registers

- **Program Counter (PC) - points to the next instruction to be executed (24 bits).**
- **General Purpose Registers - D0 through D7**
  - **Called "general purpose" because registers can each perform the same range of functions.**
  - **32 bits wide, but can be divided into 2 words or 4 bytes.**
  - **Bits in the data register have an arbitrary meaning; e.g., two's complement number, unsigned integer, or ASCII characters.**
  - **Word operations applied to these registers can only use the low order 16 bits ($d_{15}\ldots d_0$).**
  - **Byte operations applied to these registers can only use the low order 8 bits ($d_7\ldots d_0$).**
- **Address Registers - A0 through A7**
  - **Called "address registers" because they are always used to store the address of a memory location. 32 bits wide, but cannot be subdivided.**
  - **A0 through A6 can be used as you see fit; however, A7 is the stack pointer which is needed to keep track of subroutine return addresses. Therefore, you should not use A7 explicitly.**
- **CCR Register Contains the following flags:**

X eXtend flag (similar to the carry flag)
N - Negative flag - true if first bit 1 (sign bit or MSB of result is = 1)
Z Zero flag - true if all bits 0 (result is equal to zero).
V oVerflow flag (2's complement overflow)
C - Carry flag (carry out bit from an arithmetic operation).

Certain operations effect all bits; e.g., arithmetic. Certain operations effect only some of the bits (e.g., Logical operations do not effect overflow or carry). Certain operations do not effect any of the bits (e.g., exchange registers).

Computer Instruction Set Architecture (ISA) & Assembly Language

- **Instruction Set Architecture (ISA) of the Microprocessor:**

  Assembly language programmer's view of the processor.

- **Machine Code:**

  CPU language comprised of computer instructions that controls the primitive operations on binary data within the computer, including:

  - **Data movement and copying instructions**
  - **Arithmetic operations (e.g., addition and subtraction);**
  - **Logic instructions: AND, OR, XOR, shift operations, etc.**
  - **Control instructions: Jumps, Branching,**

- **Assembly Language:**

  Human-readable representation of the binary code executed by the computer.

Computer Organization Layers

- The computer may be organized into the following layers:
  - Application level language
  - High level language
  - Low level language
  - Hardware - may include microcode.
- Consider the case of a word processing program:
  - The high level commands:
    - (save, undo, bold, center, etc.) represent the application level language.
  - The high level language might be:
    - Pascal, C/C++ or Java.
  - The low level language might be:
    - 68000 or Intel x86 assembler or the proper assembly language for the CPU in use.

Basic Assembly Program Structure

- Assembly language is made up of two types of statements:
  - Executable Instruction:

    One of the processor's valid instructions which can be translated into machine code form by the assembler.

  - Assembler Directive:

    Inform the assembler about the program and the environment and cannot be translated into machine code.

    - Link symbolic names to actual values.
    - Set up pre-defined constants.
    - Allocate storage for data in memory.
    - Control the assembly process.

Assembler Directives: EQU Directive

- The equate directive, EQU simply links a name to a value in order to make a program easier to read. It does not reserve space in memory.   For example:

```
BACK_SP      EQU   $08
CAR_RET      EQU   $0D
```

- The EQU directive may include expressions as well as litrals provided all elements of the expression have already been defined:

```
Length       EQU   30
Width        EQU   25
Area         EQU   Length*Width
```

Assembler Directives: DC Directive

- This directive *defines a constant* and is qualified by:

  .B - to indicate a byte, 8 bits

  .W - to indicate a word, 16 bits

  .L - to indicate a long word, 32 bits

- The operand may consist of:
  - One or more decimal numbers;
  - One or more hexadecimal numbers denoted by a leading '$';
  - One or more binary numbers denoted by a a leading '%';
  - An ASCII string enclosed in single quotes;
  - An expression to be evaluated.
- A label in the left hand column equates the label with the first address (word).
- The constant is loaded into memory at the current location.

Assembler Directives: DS Directive

- The *define storage* directive reserves a storage location in memory but does not store any information.
- The directive may be qualified by '.B', '.W' or '.L' to indicate bytes, words or long words.
- A operand specifies the number of such quantities to reserve in decimal or hex.
- The optional label equates to the address of the first word of storage.
- Example:

```
         ORG     $1000        Starting address
FIRST    DS.B    4            Reserve 4 bytes
SECOND   DS.W    4            Reserve 4 words
THIRD    DS.L    4            Reserve 4 long words
TABLE    DS.W    $10          Reserve 16 words
```

Assembler Directives: ORG, END Directives

- The origin directive sets up the value of the location counter that tracks where the next item will be stored in memory;
  - May be located anywhere in the program.
  - Example:

```
          ORG    $00001000      Starting address
FIRST     DS.B   4              Reserve 4 bytes
          ORG    $00001100      Change the memory location
SECOND    DS.W   4              Reserve 4 words
```

- The end directive indicates that the end of the code has been reached.
  - Optionally specifies the place at which to start execution;

    e.g., END $400.

Basic Characteristics of 68000 Assembly Language

- An assembly language program line or statement is comprised of the following 4 columns:

1 Optional label which must begin in column 1

2 An instruction;

  - These are the actual instructions themselves, such as MOVE, ADD, etc.
  - Opcode fields : The suffixes `.B', `.W', and `.L' denote a byte, word, and long-word operation, respectively. If not specified, the default is word size (.W).
  - Basic addressing modes

Dn	data register
An	address register
#n	constant or immediate
n	contents of memory location

3 Its operand or operands.

4 An optional comment field.

Basic Characteristics of 68000 Assembly Language

- A line beginning with an asterisk * in the first column is a comment and is totally ignored by the assembler.
- Number systems are represented as follows:
  - A number without any prefix is *decimal.*
  - A number with a leading '$' is *hex.*
  - A number with a leading '%' is *binary.*
- Enclosing a string in quotes represents a sequence of ASCII characters.
- At least one space is required to separate the label and comment field from the instruction; but additional spaces are added for readability.
- The following data sizes apply:
  - Byte - 8 bits
  - Word - 16 bits (default operand size for most instructions).
  - Long word - 32 bits

Some Basic Assembly Instructions

Instruction	Operation Performed
MOVE D0,Q	Copy the contents of register D0 to memory location Q.
MOVE Q,D0	Copy the contents of memory location Q to register D0.
MOVE #Q,D0	Copy the number Q to register D0
ADD Q,D0	Add the contents of memory location Q to register D0 and put the result in D0.
ADD D0,Q	Add the contents of memory location Q to register D0 and put the results in memory location Q.
CLR Q	Set the content of memory location Q to zero.
CMP Q,D0	Subtract the contents of memory location Q from the contents of register D0 in order to set up the CCR. Discard the result
CMP #Q,DO	Subtract the number Q from the contents of register D0 in order to set up the CCR. Discard the result.
BEQ N	Branch to N if the result of the last operation yielded 0.
BNE N	Branch to N if operands of the last comparison were not equal.
BRA N	Always branch to location N.

68000 Operand Size and Storage in Memory

- The 68000 uses the following suffixes to identify the size of the instruction's operands:

.B	one byte
.W	word (2 bytes)
.L	long word (4 bytes)

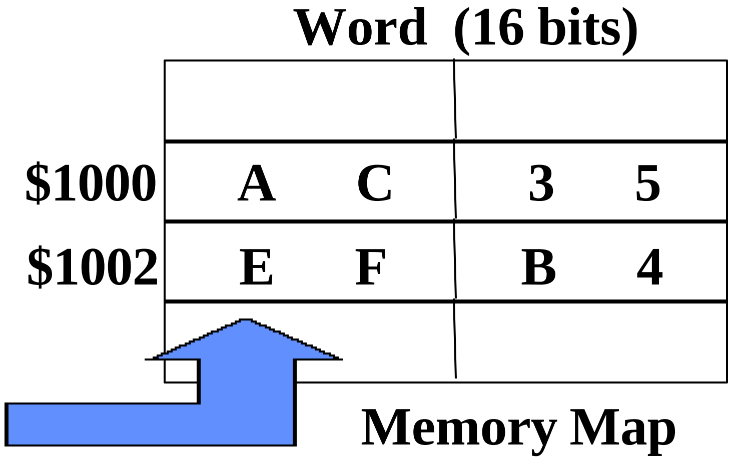
When no suffix is specified, then most instructions assume .W

- 68000 memory is byte-addressed; however, all word and long word operands in memory must start at an even address. For this reason the preferred memory map for 68000 assembly programs show a single word (two bytes) in each row.

- When storing values in memory:

  The most significant byte is stored at the first address location followed by the remaining bytes

  Example: Store $AC 35 EF B4 at memory address $1000



	Word (16 bits)	
$1000	A C	3 5
$1002	E F	B 4

Memory Map

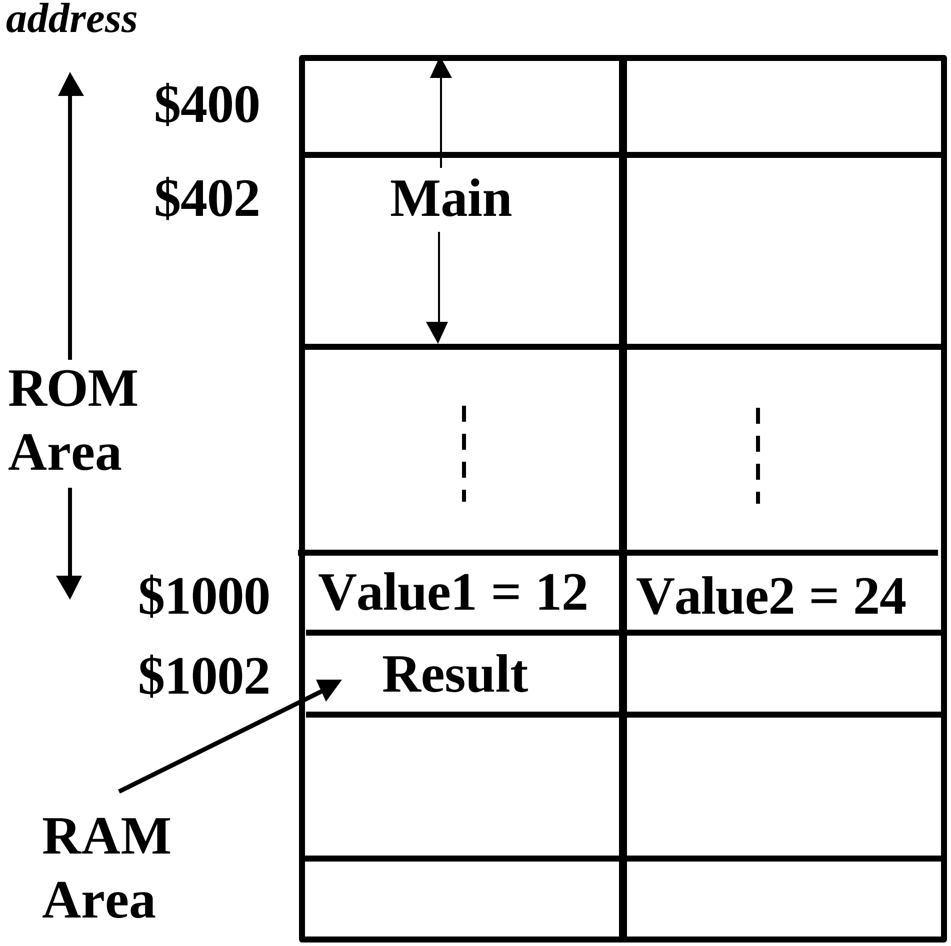
A Simple Motorola 68000 Assembly Language Program Example

- The following assembly language program adds together the two 8-bit numbers stored in the memory locations called Value1 and Value2, and deposits the sum in Result.   Result = Value1 + Value2

```
          ORG       $400          Start of program area
 Main     CLR       D0            Clear D0
          CLR       D1            Clear D1
          MOVE.B    Value1,D0     Copy Value1 to low byte of D0
          MOVE.B    Value2,D1     Copy Value2 to low byte of D1
          ADD.B     D0,D1         Add Value1 + Value2   result in D1
          MOVE.B    D1,Result     Store Result in memory
          STOP      #$2700        Stop execution
          ORG       $1000         Start of data area
Value1    DC.B      12            Store 12 in memory  for Value1
Value2    DC.B      24            Store 24 in memory for Value2
Result    DS.B      1             Reserve a memory byte for Result
          END       $400          End of program and entry point
```

Memory Map and Register Usage For Example

Memory Map



address

$400

$402 Main

ROM Area

$1000 Value1 = 12 Value2 = 24

$1002 Result

RAM Area

Register Usage

D0			Value1
D1			Value2
D2			
D3			
D4			
D5			
D6			
D7			

A0			
A1			
A2			
A3			
A4			
A5			
A6			

Example: Sum Using A Loop

- **Perform the sum 1 + 2 + 3 + … + 10 by using a loop, i.e.**

```
TOTAL := 0;
FOR COUNTER := 1 TO 10 DO
    TOTAL := TOTAL + COUNTER;
```

- **This can be accomplished by the following 68000 Assembler code:**

```
        ORG       $400        Start of program area
        CLR       D1          Set the total initially to 0
        MOVE.B    #1,D0       Initialize the counter to 1
Next    ADD.B     D0,D1       Add the counter to the total
        ADD.B     #1,D0       Increment the counter
        CMP.B     #11,D0      Check if loop is done
        BNE       Next        Go back for another round  if not done
        STOP      #$2700      Stop execution
        END       $400        Program terminator and entry point
```