Assembler Directives

An *assembler directive* is a message to the assembler that tells the assembler something it needs to know in order to carry out the assembly process; for example, an assemble directive tess the assembler where a program is to be located in memory. We are going to use the following directives in this course:

<label>	**EQU**	**<value>**	Equate
	ORG	**<value>**	Origin
<label>	**DC**	**<value>**	Define constant
<label>	**DS**	**<value>**	Define storage
	END	**<value>**	End of assembly language program and "starting address" for execution

In each case, the term <label> indicates a user-defined label (i.e., symbolic name) that must start in column 1 of the program, and <value> indicates a value that must be supplied by the programmer (this may be a number, or a symbolic name that has a value).

Equate: The **EQU** assembler directive simply equates a symbolic name to a numeric value. Consider:

```
Sunday      EQU 1
Monday      EQU 2
```

The assembler substitutes the equated value for the symbolic name; for example, if you write the instruction **ADD.B #Sunday,D2**, the assembler treats it as if it were **ADD.B #1,D2**.

You could also write

```
Sunday      EQU 1
Monday      EQU Sunday + 1
```

In this case, the assembler evaluates "Sunday + 1" as 1 + 1 and assigns the value 2 to the symbolic name "Monday".

Do not think that the EQU directive creates variables or constant. It doesn't and it has no effect on the code generated by the program. This directive simply allows you to make a name equivalent to its value (i.e., it's a form of short hand).

Origin : The *origin directive* tells the assembler where to load instructions and data into memory. The 68000 reserves the first 1024 bytes of memory for exception vectors. Your programs will start at location 1024; that is, you should begin your program with **ORG 1024** or **ORG $400** (remember that $1024 = 400_{16}$).

Define Constant : The *define constant* assembler directive allows you to put a data value in *memory at the time that the program is first loaded*. The **DC** directive takes the suffix **.B**, **.W**, or **.L**. You can put several values on one line (each value is separated by a comma). The optional *label field* is given the address of the first location in memory allocated to the **DC** function. Consider the example:

```
            ORG     $2000           Locate data here
Val1        DC.B    20,34           Store 20 and 34 in consecutive bytes
Val2        DC.L    20
```

Me DC.B 'Alan Clements'

The effect of this code is to store the value $14 in location $2000, $22 in location $2001, $00000014 in locations $2002, $2003, $2004, $2005. Remember that a 32-bit longword takes four bytes of memory. The ASCII string „Alan Clements" is stored in bytes $2006 to $2012.

If you write **MOVE.B Val2,D2**, the assembler translates it as **MOVE.B $2002,D2**. When this instruction is executed, data register D2 is loaded with the contents of memory location $2002. The value loaded into D2 might be 20. *Might be??* Yes, might be, because another instruction might modify the contents of **Val2**. By the way, if you execute **MOVE.B Me,D0**, data register D0 would be loaded with $41 (the ASCII code for „A"). However, if you execute **MOVE.W Me,D0**, data register D0 would be loaded with $416C (the ASCII code for „Al").

Define Storage : The *define storage* directive is used to reserve one or more memory locations. This directive is similar to the Pascal **type** declaration. Consider:

Result	**DS.B 1**	**Save a byte for Result**
Table	**DS.W 10**	**Save 10 words (20 bytes) for Table**
Point	**DS.L 1**	**Save 1 longword (4 bytes) for Point**

We will put these two fragments of assembly language together and assemble them using the X68K command (X68K is the Teesside 68K cross-assembler that runs under DOS on a PC). The following is part of the listing file produced by the assembler. The second column contains memory addresses and the third column contains the data loaded into these addresses.

```
2                                   ORG  $2000          ;Locate data here
00002000
3          1422          VAL1:      DC.B 20,34
00002000
4          00000014      VAL2:      DC.L 20
00002002
5          416C616E2043  ME:        DC.B 'Alan
00002006                                  Clements'
           6C656D656E74
           73
6          00000001      RESULT:    DS.B 1          ;Save a byte for Result
00002013
7          00000014      TABLE:     DS.W 10         ;Save 10 words (20 bytes) for
00002014                                             Table
8          00000004      POINT:     DS.L 1          ;Save 1 longword (4 bytes) for
00002028                                             Point
```

ALIGN: The .ALIGN directive advances the current location counter to the next specified "boundary."

Syntax

.ALIGN [*boundary*]

Parameters

boundary

An integer value for the byte boundary to which you want to advance the location counter. The Assembler advances the location counter to that boundary. Permissible values must be a power of 2 and can range from one to 4096. The default value is 8 (double word aligned).

ALLOW: The .ALLOW directive tells the Assembler to temporarily allow PA-RISC features from a higher version level of the PA-RISC architecture. The .ALLOW directive also tells the Assembler to temporarily allow implementation-specific features in the assembly source file.

Syntax

.ALLOW 1.1

Lines of source code

.ALLOW

Parameters

Title not available (Parameters)

1.1

Allows PA-RISC 1.1 features.

2.0

Allows PA-RISC 2.0 features.

relocatable object file marked as a PA-RISC 1.1 architecture version.

CALL:

The .CALL directive marks the next branch statement as a procedure call, and permits you to describe the location of arguments and the function return result.

Syntax

.CALL [*argument_description* [*argument_description*]...]

Parameters

argument_ description

Allows you to communicate to the linker the types of registers used to pass floating point arguments and receive floating point return results in the succeeding procedure call. Similarly, this information can be communicated in the .EXPORT directive.

The linker requires this information because the runtime architecture allows floating point arguments and return values to reside in either general registers or floating point registers, depending on source language convention. At link time, the linker ensures that both the caller and called procedure agree on argument location. If not, the linker may insert code to relocate the

arguments (or return result) before control is transferred to the called procedure or a procedure return is completed.

You can use up to 5 *argument-descriptions* in the .CALL directive; one for each of the four arguments that may be passed in registers (arg0-arg3), and one for a return value (ret0).

COMM: The .COMM directive makes a storage request for a specified number of bytes.

Syntax

label .COMM [*num_bytes*]

Parameters

label

Labels the location of the reserved storage.

num_bytes

An integer value for the number of bytes you want to reserve. The Assembler uses a default value of 4 if the .COMM directive lacks a *num_bytes* parameter. Permissible values range from one to 0x3FFFFFFF.

Discussion

The .COMM directive declares a block of storage that can be thought of as a *common block*. You must label every .COMM directive. The linker associates the *label* with the subspace in which the .COMM directive is declared and allocates the necessary storage within that subspace. .COMM always allocates its space in the BSS subspace of the $PRIVATE$ space. If the label of a .COMM directive appears in several object modules, the linker uses the maximum size specified in any module when it allocates the necessary storage in the current subspace.

END: The .END directive terminates an assembly language program.

Syntax

.END

Discussion

This directive is the last statement in an assembly language program. If a source file lacks an .END directive, the Assembler terminates the program when it encounters the end of the file.

The .END directive terminates an assembly language program.

Syntax

.END

Discussion

This directive is the last statement in an assembly language program. If a source file lacks an .END directive, the Assembler terminates the program when it encounters the end of the file.

ENDM : The .ENDM directive marks the end of a macro definition. The macro definition is entered into the macro table and the remaining source lines are read in and assembled. An .ENDM directive must always accompany a .MACRO directive.

Syntax

.ENDM

Example

This example defines the macro QUADL; it aligns the data specified in the macro parameters on quad word boundaries. The .ENDM directive delimits the end of the definition of QUADL.

```
QUADL  .MACRO  WD1,WD2,WD3,WD4
     .ALIGN  16
     .WORD   WD1
     .ALIGN  16
     .WORD   WD2
     .ALIGN  16
     .WORD   WD3
     .ALIGN  16
     .WORD   WD4
     .ENDM
```

Source : http://nprcet.org/e%20content/Misc/e-Learning/IT/IV%20Sem/CS%202252-Microprocessors%20and%20Microcontrollers.pdf