



# Directives

R. Ferrero, P. Bernardi

Politecnico di Torino

Dipartimento di Automatica e Informatica (DAUIN)

Torino - Italy

This work is licensed under the Creative Commons (CC BY-SA) License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-sa/3.0/>



# Instruction format

- A general source line is:

`{label} {operation} {;comment}`

- `operation` may be:

- an instruction
- a directive
- a pseudo-instruction

*labels must start at the beginning of the line*

*The instructions, directives, and pseudo-instructions must be preceded by a white space, either a tab or any number of spaces*

# Common directives

AREA	Defines a block of code or data
RN	Can be used to associate a register with a name
EQU	Equates a symbol to a numeric constant
ENTRY	Declares an entry point to your program
DCB, DCW, DCD	Allocates memory and specifies initial runtime contents
ALIGN	Aligns data or code to a particular memory boundary
SPACE	Reserves a zeroed block of memory of a particular size
LTORG	Assigns the starting point of a literal pool
END:	Designates the end of a source file

# Sections of data and code

- The IDE tools need to be told how to treat all the different parts of a program
  - Data sections,
  - Program sections,
  - Blocks of coefficients, etc.
- These sections, which are indivisible and named, then get manipulated by the linker and ultimately end up in the correct type of memory in a system.
  - The data, which could be read-write information, could get stored in RAM,
  - The program code which might end up in Flash memory.

## Sections of data and code (II)

- Normally you will have separate sections for your program and your data, especially in larger programs.
  - The code must have at least one AREA directive in it, which is usually found in the first few lines of a program
  - Blocks of coefficients or tables can be placed in a section of their own.

# Sections of data and code (III)

- `AREA sectionName {, attr} {, attr}...`
- If *sectionName* starts with a number, it must be enclosed in bars  
e.g. `|1_DataArea|`
- `|.text|` is used by the C compiler
- At least one AREA directive is mandatory
- Example: `AREA Example, CODE, READONLY`

# Section attributes

- `CODE`: the section contains machine code
- `DATA`: the section contains data
- `READONLY`: the section can be placed in read-only memory
- `READWRITE`: the section can be placed in read-write memory
- `ALIGN = expr`: the section is aligned on a  $2^{*expr*}$ -byte boundary

# Register names

- r0-r15 or R0-R15
- a1-a4 or r0-r3
- r13, R13, sp, SP
- r14, R14, lr, LR
- r15, R15, pc, PC
- You can assign other names with RN:

*name RN registerIndex*

- E.g. `coeff1 RN 8`



# Declaring constants

- The `EQU` directive gives a symbolic name to a numeric constant:

*name EQU expression*

- Advantages:
  - readability
  - easiness in updating the value through the code

# Numbers

- You can express numbers in any base:
  - decimal: e.g. 123
  - hexadecimal: e.g. 0x3F
  - other bases in the format n\_xxx where
    - n is the base
    - xxx is the number in that base

# Constant allocation in code memory

`{label} DCxx expr{,expr}...`

- The available directives are:
  - DCB: define constant byte
  - DCW: define constant half-word
  - DCWU: define constant half-word unaligned
  - DCD: define constant word
  - DCDU: define constant word unaligned
- *expr* is:
  - a numeric expression in the proper range
  - a string (with DCB only)

# DCB

```
myData    DCB 65, 0x73, 8_163  
          DCB "embly"
```

Address	Value	Octal	Hex	ASCII
0x000000D2	65	101	41	A
0x000000D3	115	163	73	s
0x000000D4	115	163	73	s
0x000000D5	101	145	65	e
0x000000D6	109	155	6D	m
0x000000D7	98	142	62	b
0x000000D8	108	154	6C	l
0x000000D9	121	171	79	y

# DCW

myData    DCB   65, 0x73, 8\_163  
            DCW   0x626D, 0x796C

Address	Value	Octal	Hex	ASCII
0x000000D2	65	101	41	A
0x000000D3	115	163	73	s
0x000000D4	115	163	73	s
0x000000D5	0	0	0	NUL
0x000000D6	109	155	6D	m
0x000000D7	98	142	62	b
0x000000D8	108	154	6C	l
0x000000D9	121	171	79	y

# DCWU

myData    DCB 65, 0x73, 8\_163  
           DCWU 0x626D, 0x796C

Address	Value	Octal	Hex	ASCII
0x000000D2	65	101	41	A
0x000000D3	115	163	73	s
0x000000D4	115	163	73	s
0x000000D5	109	155	6D	m
0x000000D6	98	142	62	b
0x000000D7	108	154	6C	l
0x000000D8	121	171	79	y

# DCD

```
myData    DCB  65, 0x73, 8_163  
          DCD 0x796C626D
```

Address	Value	Octal	Hex	ASCII
0x000000D2	65	101	41	A
0x000000D3	115	163	73	s
0x000000D4	115	163	73	s
0x000000D5	0	0	0	NUL
0x000000D6	0	0	0	NUL
0x000000D7	0	0	0	NUL
0x000000D8	109	155	6D	m
0x000000D9	98	142	62	b
0x000000DA	108	154	6C	l
0x000000DB	121	171	79	y

# DCDU

```
myData    DCB  65, 0x73, 8_163
          DCDU 0x796C626D
```

Address	Value	Octal	Hex	ASCII
0x000000D2	65	101	41	A
0x000000D3	115	163	73	s
0x000000D4	115	163	73	s
0x000000D5	109	155	6D	m
0x000000D6	98	142	62	b
0x000000D7	108	154	6C	l
0x000000D8	121	171	79	y



# Align

- The `ALIGN` directive aligns the current location to a specified boundary by padding with zeros:

`ALIGN {expr{, offset}}`

- The current location is aligned to the next address of the form

$$n * \textit{expr} + \textit{offset}$$

- If *expr* is not specified, `ALIGN` sets the current location to the next word boundary.
- Example: The `ADR` Thumb pseudo-instruction can only load addresses that are word aligned, but a label within Thumb code might not be word aligned. Use `ALIGN 4` to ensure four-byte alignment of an address within Thumb code.

# Align expr

```
myData    DCB  65  
          ALIGN 2  
          DCB 115
```

Address	Value	Octal	Hex	ASCII
0x000000D4	65	101	41	A
0x000000D5	0	0	0	NUL
0x000000D6	115	163	73	s

# Align expr

```
myData    DCB  65  
          ALIGN 4  
          DCB 115
```

Address	Value	Octal	Hex	ASCII
0x000000D4	65	101	41	A
0x000000D5	0	0	0	NUL
0x000000D6	0	0	0	NUL
0x000000D7	0	0	0	NUL
0x000000D8	115	163	73	s

# Reserving a block of memory

- The SPACE directive reserves a zeroed block of memory:

`{label} SPACE expr`

- `expr` is the number of bytes to reserve

- Example:

```
long_var SPACE 8
```

# Ending the source file

- The `END` directive tells the assembler that the current location is the end of the source file:

`END`