Common Directives

AREA: Definisce uno spazio di codice/data
RN: Permette di dare un *alias* ad un registro.
EQU: Permette di dare un *alias* ad una costante.
ENTRY: Definisce un punto di entrata nel programma: una specie di *main()*
DCB,DCW,DCD: Alloca della memoriae inizializza i contenuti(B=8 bit,W=16 bit,D=32bit). L'assembler scrive in memoria tali costanti.
ALIGN: Allinea un blocco di dati/codice ad un particolare *memory boundary*
SPACE: Inizializza a zero un determinato blocco di memoria
LTORG: Assegna il punto di inizio di una *literal pool*
END: Indica la fine del file sorgente

Sections of data and code

Ci sono diverse sezioni:

- **Data:** simile al *.data* del **MIPS**
- **Program:** simile al *.code* del **MIPS**. Il compilatore C usa |*.text*|
- **Blocchi di coefficienti:** usata per salvare blocchi di costanti

Queste sezioni vengono create con la *direttiva* **AREA**.
Devono esserci sempre delle sezioni **data** e **codice**
Sintassi: **AREA** *sectionName {,attr} {,attr}*

Section attributes

CODE: La sezione contiene codice
DATA: La sezione contiene dati
NOINIT: La sezione non verrà inizializzata
READONLY: Questa sezione può essere piazzata in una memoria Readonly(ad esempio una memoria flash)
READWRITE: Questa sezione può essere piazzata in una memoria Readwrite
ALIGN = expr: La sezione viene allineata su un boundary di $2^{\text{expr}}\,\text{byte}$

Register names and costants alias

Per dare un alias ad un registro: *name* **RN** *registerIndex*
ad esempio: **coeff1** RN 8 fa sì che R8 possa venir chiamato scrivendo **coeff1**
Inoltre, r13/R13 di default ha come alias sp/SP ,r14/R14 ha lr/LR e r15/R15 ha pc/PC

In modo analogo, si dichiarano delle costanti:
name **EQU** *expr*
ad esempio: pi EQU 3

Numbers

Possono essere rappresentati in qualsiasi base:

- **decimale:** 123
- **hex:** 0x3F
- **altre basi:** n_xxx dove n è la base, xxx il numero

Defining costants

Si dichiarano con la forma:
{label} **DCxx** expr{,expr}
dove xx:

- **B:** byte
- **W:** half-word
- **WU:** half-word non allineata
- **D:** word
- **DU:** word non allineata

DCB Example

```
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

Notare come l'indirizzo aumenta di byte in byte quando normalmente aumenta di 4 byte alla volta

DCW Example

```
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

Notare due cose:

1. *Il valore a 0xD5 è NUL in quanto DCW dichiara costanti allineati a 2 byte, essendo un halfword composto da 2 byte e dunque anche l'indirizzo deve essere multiplo di 2 byte*
2. *La endianess delle halfword. Il LSB và nell'indirizzo più basso mentre il MSB nell'indirizzo più alto*

DCWU Example

```
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

In questo caso non ci sarà nessun NUL in quanto forzo il non allineamento della memoria

DCD Example

```
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

Stessa cosa di sopra. Il primo valore di DCD viene piazzato al primo indirizzo multiplo di 4 byte

disponibile dal momento che DCD allinea a 4 byte

DCDU Example

```
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

Come sopra ma l'allineamento non viene forzato.

Align

Permette di allineare la locazione corrente al boundary specificato inserendo padding con 0.
Allinea di default a 4 byte(una word) ma è possibile specificare, attraverso la seguente sintassi
ALIGN *{expr{,offset}}* dove
expr: è il numero di byte a cui bisogna allineare
offset: offset da aggiungere

Align simple example

```
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

Dopo DCB 65, la memoria viene allineata a multipli di 2 byte

Align example with offset

```
myData   DCB 65
         ALIGN 4, 3
         DCB 115
```

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

Allineo a multipli di 4 byte prima di uno spiazzamento di 3 byte. In questo caso cerco il primo indirizzo multiplo di 4 byte e poi aggiungo 3 byte di offset.

SPACE directive

Simile alla direttiva analoga del **MIPS**.
{label} **SPACE** *expr* permette di inizializzare *expr* byte a 0 un'area di memoria **read-write**

END directive

Per indicare la fine del file sorgente