Per poter eseguire del codice in *C*, dobbiamo prima compilarlo.

Durante questo processo si passa per diverse fasi, tra cui quella di traduzione ad un altro linguaggio:

assembly.

Questo processo è necessario per creare un file eseguibile dal sistema operativo

Oggi proveremo a effettuare questo processo «manualmente» utilizzando come linguaggio assembly: m68k







Github questa lezione

REGISTRI

Conoscete le <mark>variabili?</mark> Bene, in m68k esistono, sono di un <u>numero limitato</u> e si chiamano <mark>registri</mark>

18 registri utilizzabili, 8 data e 8 address

d0,d1,d2,d3,d4,d5,d6,d7 a1,a2,a3,a4,a5,a6,a7

Noi useremo solo i registri data, che sono quelli che iniziano con la lettera d



MOME

<u>Copia</u> il contenuto del primo operando nel secondo operando



<secondo operando>

move <primo operando>, <secondo operando>

move #100, d0

Cosa succede nell'esempio?

Mettiamo il numero 100 dentro il registro do

move <primo operando>, <secondo operando>

move d1, d0

Cosa succede nell'esempio?

Copiamo il

contenuto del registro d1 dentro il registro d0

Il secondo operando non può essere mai un numero

ADD

Effettua la somma di due valori e salva il risultato nel secondo

add <primo operando>, <secondo operando>

add #100, d0

Cosa succede nell'esempio?

Sommiamo 100 con il contenuto di d0 e inseriamo il risultato dentro il registro d0

add <primo operando>, <secondo operando>

add d1, d0

$$d0 = d1 + d0$$

Cosa succede nell'esempio?

Sommiamo i contenuti dei due registri e inseriamo il risultato dentro il registro do

Il secondo operando non può essere mai un numero

SUB

Effettua la sottrazione del secondo valore meno il primo e salva il risultato nel secondo

Risultato <secondo operando>

sub <primo operando>, <secondo operando>

Cosa succede nell'esempio?

Sottraiamo il contenuto di do con 100 e inseriamo il risultato dentro il registro do

sub <primo operando>, <secondo operando>

$$d0 = d0 - d1$$

Cosa succede nell'esempio?

Sottraiamo i contenuti dei due registri e inseriamo il risultato dentro il registro do

Il secondo operando non può essere mai un numero

MULS

Moltiplica il secondo registro per il primo valore/registro e salva il risultato nel secondo registro

muls <primo operando>, <secondo operando>

muls #100, d0

d0 = d0 ***** 100

Cosa succede nell'esempio?

Moltiplichiamo il contenuto di do con 100 e inseriamo il risultato dentro il registro do

muls <primo operando>, <secondo operando>

muls d1, d0

d0 = d0 * d1

Cosa succede nell'esempio?

Moltiplichiamo i contenuti dei due registri e inseriamo il risultato dentro il registro do

Il secondo operando non può essere mai un numero

CMP

Ha il compito di comparare due operandi Viene sempre utilizzato con i comandi di branch

Comando	Logicamente	Acronimo
beq	b == a	Branch equal
	b != a	Branch not equal
blt	b < a	Branch less than
	b <= a	Branch less or equal
bgt	b > a	Branch greater than
	b >= a	Branch greater or equal

COMANDI DI BRANCH



Cosa succede nell'esempio?

Se d1 == d0 allora facciamo un salto alla label chiamata sezione2

I comandi a destra per funzionare devono essere seguiti dalla label di dove il programma deve «saltare»

comando < label>

Comando	Logicamente	Acronimo
beq	b == a	Branch equal
	b != a	Branch not equal
blt	b < a	Branch less than
	b <= a	Branch less or equal
bgt	b > a	Branch greater than
	b >= a	Branch greater or equal



LABEL

cmp <d0>, <d1>
beq sezione2

sezione2:

Cosa succede nell'esempio?

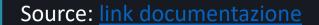
Se d1 == d0 allora facciamo un salto alla label chiamata sezione2

Semantica label per fare il salto:

nomeLabel:

Da ricordare di mettere i due punti alla fine

Comando	Logicamente	Acronimo
beq	b == a	Branch equal
	b != a	Branch not equal
blt	b < a	Branch less than
	b <= a	Branch less or equal
bgt	b > a	Branch greater than
	b >= a	Branch greater or equal



BRA

E' un salto incondizionato, va direttamente alla label scritta senza fare ulteriori controlli

bra <label>

Cosa succede nell'esempio?

<label>:

Facciamo un salto incondizionato verso la label con lo stesso nome

