Università degli Studi di Milano
Corso di Laurea in Informatica, A.A. 2018-2019

Direttive MIPS per i dati

Turno A
Nicola Basilico
Dipartimento di Informatica
Via Celoria 18 - 20133 Milano (MI)
Ufficio 4008
nicola.basilico@unimi.it
+39 02.503.16289

Turno B
Marco Tarini
Dipartimento di Informatica
Via Celoria 18 - 20133 Milano (MI)
Ufficio 4010
marco.tarini@unimi.it
+39 02.503.16217

Sezioni del programma

- sezione `.text` del programma: sequenza di istruzioni
  - 1 istruzione diventa 4 byte in RAM (sono istruz MIPS!)
  - eccezione: alcune pseudo-istruzioni vengono tradotte dall'assember in 2 istruzioni = 8 byte
- sezione `.data` del programma: sequenza di numeri
  - 1 numero diventa 4 byte in RAM (sono word!)
  - almeno, di default
- in entrambi i casi:
  - i byte sono scritti in RAM in sequenza: un dato / istruzione dopo l'altra
  - posso usare le label per recuperare l'indirizzo in RAM a cui sono finiti

Esempio

- Un programma che non fa nulla
  ma con 5x4 = 20 byte di dati numerici
  in RAM

```
.data
12 13 23
54 11
.text
```

Esempio

1. I dati di questo programma rappresentano le tre dimensioni di una scatola (in cm)
   – altezza, lunghezza & profondità
2. Copiamo i tre dati dalla RAM in tre registri
   – scegliamo $s1 $s2 $s3
3. Moltiplichiamoli fra loro per trovare il volume (in cm^3)
   – risultato in $s4

```
.data
misure:
32 20 55

.text
la $t0 misure
lw $s1  ($t0)
lw $s2 4($t0)
lw $s3 8($t0)


mul $t1 $s1 $s2
mul $s4 $t1 $s3
```

Esempio: note

- Per leggere il dato dall'indirizzo indicato dall'etichetta «misure»…

```
lw $s1 (misure)
```

- Errore! in MIPS, le operazioni su memoria (come la «load word») richiedono di specificare l'indirizzo RAM come il registro che lo contiene
  - motivo: un indirizzo di 32 bits non potrebbe essere contenuto da un'istruzione MIPS da 32 bit
- Bisogna dunque prima preparare l'indirizzo in un registro (qui: $t1)

```
la $t0 misure
lw $s1  ($t0)
```

Esempio: note

- Per moltiplicare i tre numeri ed ottenere il volume

```
mul $s4 $s1 $s2 $s3
```

- Errore! in MIPS, le operazioni matematiche come multiply ammettono solo due operatori
- Occorre dunque spezzare il computo in due istruzioni, usando un registro per il valore intermedio (qui: $t1)

```
mul $t1 $s1 $s2
mul $s4 $t1 $s3
```

Specificare i dati in altri formati

- Posso cambiare il formato in cui specifico i dati con apposite «direttive»
- Ognuna di queste direttive cambia il modo in cui l'assembler interpreta i dati che scrivo di seguito
  - valgono cioè fino a contrordine
- Nota: questo è indipendente dalle etichette
  - ricordare la sitassi:
    l'etichetta termina con due-punti
    la direttiva inizia con punto

Direttive per la specifica dati

`.word`	Ogni numero che segue = un word (4 bytes)
`.half`	Ogni numero che segue = un half-word (2 bytes)
`.byte`	Ogni numero che segue = un byte
`.ascii`	Ogni *stringa* che segue, messa fra "virgolette" = 1 byte per lettera – il codice ASCII di quella lettera
`.asciiz`	Come sopra, più un ultimo ulteriore byte 0x00 per terminare la stringa
`.space n`	Lascia qui *n* byte di spazio (salta *n* byte prima di inserire il prossimo dato)
`.align n`	Lascia qui un certo numero di byte per rendere la prossima locazione di memoria divisible per 2^n

Esempio

```
.data
altezza: .word 170
peso: .word 64000  # in grammi
voti: .half 28 21 30 27 24
eta: .byte 24

.text
# copio eta in $s0
la $t1 eta
lb $s0 ($t1)

# copio il 4to voto (27) in $s1
la $t1 voti
lh $s1 6($t1)
```

Osservazione

- Combinando i meccanismi di direttive e etichette, ottengo una sintassi che somiglia superficialmente a quella di una *dichiarazione delle variabili* in un linguaggio ad alto livello (C, Java, Go…)
  - La direttiva somiglia al TIPO della variabile
  - L'etichetta somiglia all'IDENTIFICATORE della variabile
  - Il valore del dato somiglia all'INIZIALIZZAZIONE

```
peso: .word 65000
```
assembly MIPS

```
peso := int 65000 ;
```
Go

```
int peso = 65000 ;
```
C o Java

Osservazione

- La somiglianza è superficiale
- A differenza delle variabili in un linguaggio ad alto livello, i nostri dati in memoria RAM non sono associati ad alcun tipo
- Sta al programmatore (o al compilatore) MIPS usarli in modo consistente con loro semantica / tipo
- Per es
  - nulla distingue in indirizzo di memoria a cui memorizzo un array di numeri da quello in cui memorizzo in numero solo
  - posso definire 4 byte in successione, e poi usarli come un singolo word, oppure come una stringa di 4 lettere

Direttive .byte, .word, .half

- Posso esprimere i valori anche in esadecimale:

```
.word 0xABCD0000
.half 0xAB13  0xAC01
.byte 0xAF
```

- Posso esprimere i valori come numeri negativi (vengono interpretati in complemento a 2)

```
.word -1
```
equivalente a
```
.word 0xFFFFFFFF
```
equivalente a
```
.word 4294967295
```

```
.byte -1
```
equivalente a
```
.byte 0xFF
```
equivalente a
```
.byte 255
```

Direttiva .byte

- L'assembler MARS scrive valori secondo la endianness della macchina sui cui esegue

```
.byte 0xAA 0xBB 0xCC 0xDD
```

equivalente a:

```
.word 0xDDCCBBAA
```

se la macchina è little endian

La famiglia di architetture x86 è Little Endian (Intel Core i7, AMD Phenom II, FX, …).

Direttiva .space : note

- L'assembler lascia dello spazio in memoria in RAM
- Non è previsto che venga cancellata
- Se il programma **legge** da questo spazio prima di scriverci, può trovare qualsiasi valore
  - per es, il valore lasciato in quell'area di memoria da un programma eseguito in precedenza
  - Può dipendere da: il sistema operativo, quale programma è stato eseguito in precedenza
- E' detta «Memoria sporca»
- Un programma corretto **scrive** nella memoria RAM riservata in questo modo, prima di leggerla

Esempio 1

```
.data
peso: .space 4

.text

# il peso è 68250 grammi…
la $t1 peso
li $s2 68250
sw $s2 (t1)
```

Esempio 2

1. I dati del programma:
   le tre misure di una scatola (in cm) e uno misura del volume (da calcolare)
2. Copiamo le misure dalla RAM in tre registri
   - scegliamo $s1 $s2 $s3
3. Moltiplichiamoli fra loro per trovare il volume (in cm^3)
   - risultato in $s4
4. Storiamo il risultato nel dato indicato da «volume»

```
.data
misure: .word 32 20 55
volume: .space 4

.text
la $t0 misure
lw $s1  ($t0)
lw $s2 4($t0)
lw $s3 8($t0)

mul $t1 $s1 $s2
mul $s4 $t1 $s3

la $t0 volume
sw $s4 ($t0)
```

Direttive .ascii e .asciiz

```
.ascii "Derp"
```

equivalente a:

```
.byte 0x44 0x65 0x62 0x70
```

Direttive .ascii e .asciiz

```
.asciiz "Derp"
```

equivalente a:

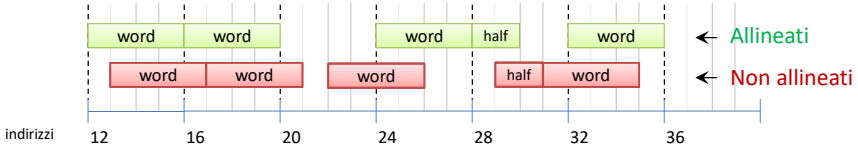
```
.byte 0x44 0x65 0x62 0x70 0x00
```

Tabella ascii

Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char
0	0	0		32	20	40	[space]	64	40	100	@	96	60	140	`
1	1	1		33	21	41	!	65	41	101	A	97	61	141	a
2	2	2		34	22	42	"	66	42	102	B	98	62	142	b
3	3	3		35	23	43	#	67	43	103	C	99	63	143	c
4	4	4		36	24	44	$	68	44	104	D	100	64	144	d
5	5	5		37	25	45	%	69	45	105	E	101	65	145	e
6	6	6		38	26	46	&	70	46	106	F	102	66	146	f
7	7	7		39	27	47	'	71	47	107	G	103	67	147	g
8	8	10		40	28	50	(	72	48	110	H	104	68	150	h
9	9	11		41	29	51	)	73	49	111	I	105	69	151	i
10	A	12		42	2A	52	*	74	4A	112	J	106	6A	152	j
11	B	13		43	2B	53	+	75	4B	113	K	107	6B	153	k
12	C	14		44	2C	54	,	76	4C	114	L	108	6C	154	l
13	D	15		45	2D	55	-	77	4D	115	M	109	6D	155	m
14	E	16		46	2E	56	.	78	4E	116	N	110	6E	156	n
15	F	17		47	2F	57	/	79	4F	117	O	111	6F	157	o
16	10	20		48	30	60	0	80	50	120	P	112	70	160	p
17	11	21		49	31	61	1	81	51	121	Q	113	71	161	q
18	12	22		50	32	62	2	82	52	122	R	114	72	162	r
19	13	23		51	33	63	3	83	53	123	S	115	73	163	s
20	14	24		52	34	64	4	84	54	124	T	116	74	164	t
21	15	25		53	35	65	5	85	55	125	U	117	75	165	u
22	16	26		54	36	66	6	86	56	126	V	118	76	166	v
23	17	27		55	37	67	7	87	57	127	W	119	77	167	w
24	18	30		56	38	70	8	88	58	130	X	120	78	170	x
25	19	31		57	39	71	9	89	59	131	Y	121	79	171	y
26	1A	32		58	3A	72	:	90	5A	132	Z	122	7A	172	z
27	1B	33		59	3B	73	;	91	5B	133	[	123	7B	173	{
28	1C	34		60	3C	74	<	92	5C	134	\	124	7C	174	|
29	1D	35		61	3D	75	=	93	5D	135	]	125	7D	175	}
30	1E	36		62	3E	76	>	94	5E	136	^	126	7E	176	~
31	1F	37		63	3F	77	?	95	5F	137	_	127	7F	177	

Allineamento dati

- L'accesso a memoria si dice allineato su *n* byte se:
  - ogni dato di dimensione *n* byte ha un è multiplo di *n*
  - *n* è una potenza di 2 (es 2, 4, 8 o 16)
- In MIPS l'accesso a word è allineato a 4:
  - il loro indirizzo deve essere multiplo di 4
  - altrimenti: viene generato un errore a runtime (una «trap»)
  - l'accesso ai singoli byte però è libero



Esercizio (da sabotatore)

- Creare un errore a run-time causato dal mancato allineamento («mis-alignment»)
  - sia su word che su half-word

Tentativo 1

```
.data
eta: .byte 24
peso: .word 64000

.text
la $t1 peso
lw $s0 ($t1)
```

Tentativo 2

```
.data
nome: .asciiz "Zorro"
peso: .word 64000

.text
la $t1 peso
lw $s0 ($t1)
```

Non viene generata la trap.
Motivo: l'assembler di MARS auto-allinea le word.

Soluzione 1

```
.data
peso: .word 65000

.data
la $t1 peso
lw $s0 1($t1)
```

Trap!
Accesso in lettura a indirizzo non divisibile per 4

Soluzione 2

```
.data
eta: .byte 19
peso: .space 4

.text
li $s0 65000
la $t1 peso
sw $s0 ($t1)
```

Trap!
Accesso in scrittura a indirizzo non divisibile per 4

Soluzione 2: fix con align

```
.data
eta: .byte 19
.align 2
peso: .space 4

.text
li $s0 64000
la $t1 peso
sw $s0 ($t1)
```

MARS: «esecuzione terminata con successo»

Direttiva .align

- Forza l'assembler a saltare il num. di byte necessario a far sì che il prossimo dato parta allineato a…
  - `.align 1` 2^1 = 2 byte
  - `.align 2` 2^2 = 4 byte -- cioè al word
  - `.align 3` 2^3 = 8 byte
- Per es, allineando a 4 bytes, verranno saltati 0, 1, 2, oppure 3 bytes (anche 0)
- In gergo: si fa «padding» per allineare i dati
- Nota: potremmo sempre calcolare noi stessi il num necessario e usare .space, ma è più comodo lasciare il tedioso compito all'assembler, usando .align