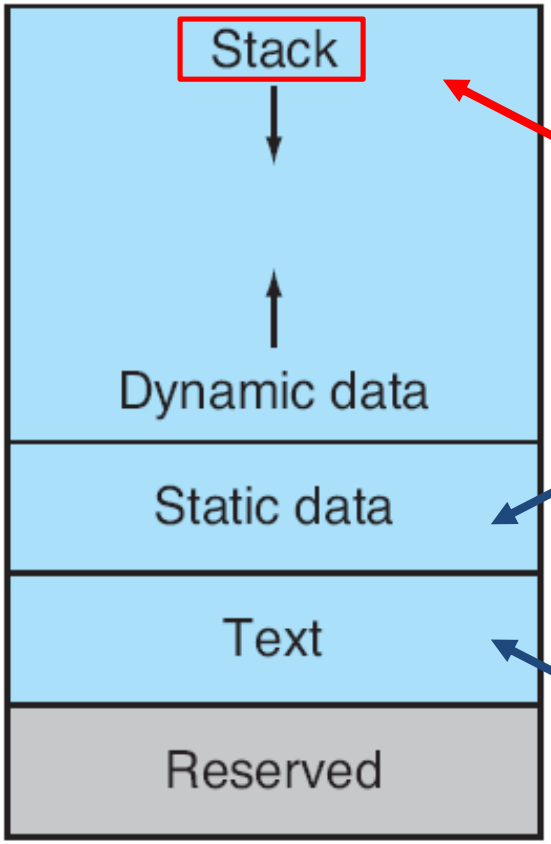
Università degli Studi di Milano
Dipartimento di Informatica “Giovanni Degli Antoni”
Corso di Laurea Triennale in Informatica

Architettura degli Elaboratori II Laboratorio

Uso dello Stack

Lo Stack



Regione nell'area dei dati dinamici che utilizziamo rispettando una regola: allocazioni e deallocazioni seguono un **coda LIFO** (last in, first out)

I dati statici usati dal nostro programma, ciò che sta dopo la direttiva `.data`

Le istruzioni del nostro programma, ciò che sta dopo la direttiva `.text`

- Il nostro programma può allocare/deallocare dati sullo stack durante l'esecuzione. **Come si fa?**

Stack Pointer

- Il registro `$sp` (stack pointer) contiene sempre l'indirizzo della parola che sta in cima allo stack. Questa parola è l'ultima ad essere stata inserita nello stack e sarà la prima ad essere rimossa.

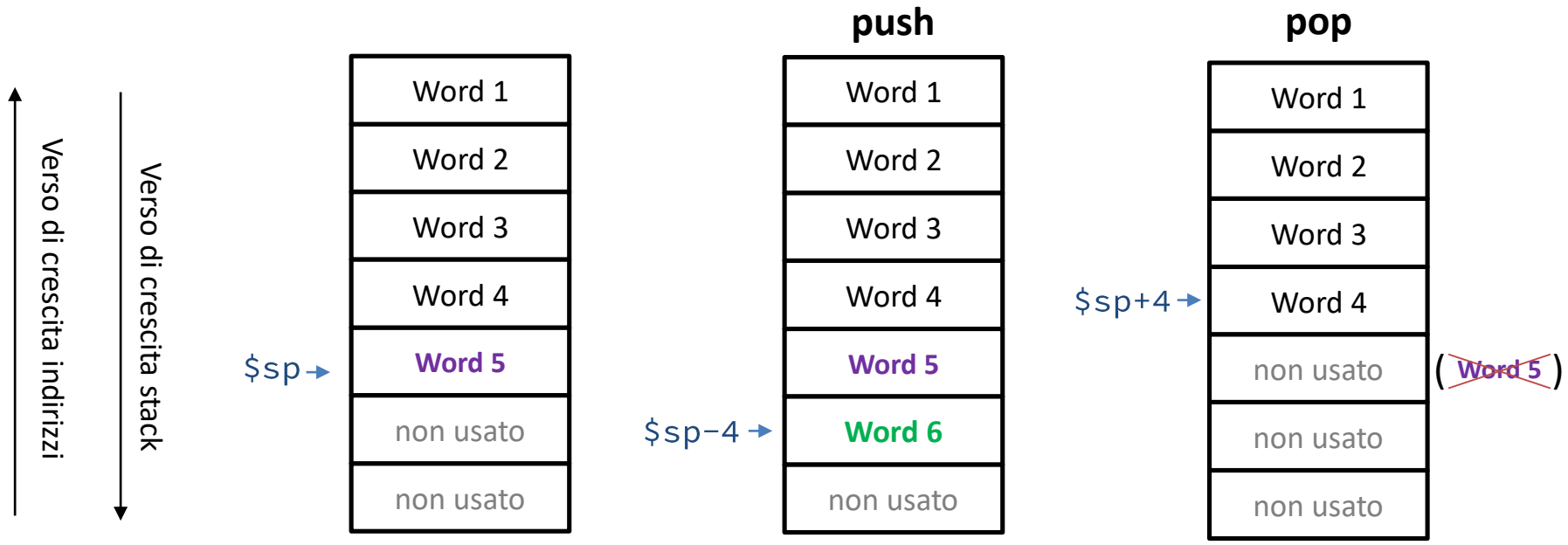
Regole per l'utilizzo dello stack

Per aggiungere un **dato** (push):

```
addi $sp $sp -4
sw $reg 0($sp)
```

Per consumare un **dato** (pop):

```
lw $reg 0($sp)
addi $sp $sp, 4
```



Register Spilling

- Quando serve salvare dati sullo stack? Una prima risposta è «Per fare **spilling** di registri»

- In generale, un programma potrebbe avere un numero di variabili maggiore rispetto al numero di registri disponibili. Non è possibile avere tutti i dati nel banco registri allo stesso momento.

- Una possibile soluzione è questa: tengo nei registri i dati di cui ho maggior bisogno (ad esempio quelli che devo usare più volte o più spesso) mentre i dati di cui non ho bisogno urgente li sposto temporaneamente in memoria.

- Trasferire variabili poco utilizzate da registri a memoria è **register spilling**.

- L'area di memoria di solito utilizzata per questa operazione è lo stack.

Register Spilling

Esempio

- Supponiamo di poter utilizzare solo i registri $t0 e $t1
- Dobbiamo calcolare il prodotto di due variabili che stanno nel segmento dati e i cui indirizzi sono identificati dalle label x e y

```
        .data
x:      .word 3
y:      .word 4

        .text
        .globl main
main:
        la $t0 x
        lw $t1 0($t0)

        add $sp $sp -4
        sw $t1 0($sp)

        la $t0 y
        lw $t1 0($t0)

        lw $t0 0($sp)
        add $sp $sp 4

        mult $t0 $t1
        mflo $t0
```

Spilling (push)

Spilling (pop)

Uso delle System Calls

System Calls

- **System call**: permette di utilizzare **servizi** la cui esecuzione è a carico del sistema operativo: tipicamente operazioni di input/output

- Ogni servizio è associato ad un codice numerico univoco (un intero)

- Come si utilizza una system call che ha codice **K**?
  - Caricare **K** nel registro `$v0`;
  - caricare gli argomenti (se necessari) nei registri `$a0, $a1, $a2, $a3, $f12` (opzionale);
  - eseguire l'istruzione `syscall`;
  - leggere eventuali valori di ritorno nei registri `$v0, $f0` (opzionale).

System Calls ‘’Canoniche’’ (SPIM)

Syscall	Codice	Argomenti	Valore di ritorno	Descrizione
print_int	1	intero da stampare in $a0	nessuno	Stampa l’intero passato in $a0
print_float	2	float da stampare in $f12	nessuno	Stampa il float passato in $f12
print_double	3	double da stampare in $f12	nessuno	Stampa il double passato in $f12
print_string	4	Indirizzo della stringa da stampare in $a0	nessuno	Stampa la stringa che sta all’indirizzo passato in $a0
read_int	5	nessuno	Intero letto in $v0	Legge un intero in input e lo scrive in $v0
read_float	6	nesuno	Float letto in $f0	Legge un float in input e lo scrive in $f0
read_double	7	nessuno	Double letto in $f0	Legge un double in input e lo scrive in $f0
read_string	8	Indirizzo nel segmento dati a cui salvare la stringa in $a0 e lunghezza in byte in $a1	nessuno	Legge una stringa di lunghezza specificata in $a1 e la scrive nel segment dati all’indirizzo specificato in $a0
sbrk	9	Numero di byte da allocare in $a0	Indirizzo del primo dei byte allocati in $v0	Accresce il segmento dati allocanto un numero di byte specificato in $a0, restituisce in $v0 l’indirizzo del primo di questi nuovi byte
exit	10	nessuno	nessuno	Termina l’esecuzione

System Calls "Apocrife" (MARS)

Syscall	Codice	Argomenti	Valore di ritorno	Descrizione
Time	30	nessuno	32 bit meno significativi del system time in $a0, 32 bit più significativi del system time in $a1	Il system time è rappresentato nel formato Unix Epoch time, cioè il numero di millisecondi trascorsi dal 1 Gennaio 1970
random int	41	Id del generatore pseudo-random in $a0	Prossimo numero pseudo random in $a0	Ad ogni chiamata restituisce un numero intero in una sequenza pseudo-random
random in range	42	Id del generatore pseudo-random in $a0, massimo intero generabile in $a1	Prossimo numero pseudo random in $a0	Ad ogni chiamata restituisce un numero intero in una sequenza pseudo-random, ogni numero sarà compreso tra 0 e il massimo passato in $a1
MessageDialog	55	Indirizzo della stringa da stampare in $a0, intero corrispondente al tipo di messaggio in $a1		Mostra una finestra di dialogo con un messaggio dato dalla stringa passata in $a0. Viene anche mostrata una icona che dipende dal tipo di messaggio passato in $a1: errore (0), info, (1), warning (2), domanda(3)
InputDialogInt	51	Indirizzo della stringa da stampare in $a0	Intero letto in $a0, stato in $a1	

Esempio

```
        .data
msg1: .asciiz "Hello world!"
msg2: .asciiz "Inserisci un intero"

        .text
        .globl main
main:

        li $v0 4,
        la $a0, msg1
        syscall

        li $v0 55
        la $a0 msg1
        li $a1 1
        syscall

        li $v0, 51
        la $a0, msg2
        syscall

        li $v0 10
        syscall
```

Stampiamo una stringa nello standard output (la console)

Stampiamo una stringa in una finestra di diaologo

Leggiamo un intero in input con una finestra di dialogo

Exit