I registri e convenzioni sul loro uso

Registro	Nome	Utilizzo
x0	zero	Costante zero
x1	ra	Return address
x2	sp	Stack pointer
x3	gp	Global pointer
x4	tp	Puntatore a thread
x8	s0 / fp	Frame pointer (il contenuto va preservato se utilizzato dalla procedura chiamata)
x10-x11	a0-a1	Passaggio di parametri nelle procedure e valori di ritorno
x12-x17	a2-a7	Passaggio di parametri nelle procedure
x5-x7 x28-x31	t0-t2 t3-t6	Registri temporanei, non salvati in caso di chiamata
x9	s1	Registri da salvare: il contenuto va preservato se utilizzati dalla procedura chiamata
x18-x27	s2-s11	

Convenzione nell'uso e salvataggio dei registri

- Chi è responsabile di salvare i registri quando si effettuano chiamate di funzioni?
  - La funzione chiamante conosce quali registri sono importanti per sé e che dovrebbero essere salvati
  - La funzione chiamata conosce quali registri userà e che dovrebbero essere salvati prima di modificarli
- Bisogna evitare le inefficienze → Minimo salvataggio dei registri:
  - La funzione chiamante potrebbe salvare tutti i registri che sono importanti per sé, anche se la procedura chiamata non li modificherà
  - La funzione chiamata potrebbe salvare tutti i registri che si appresta a modificare, anche quelli che non verranno poi utilizzati dalla procedura chiamante una volta che la procedura chiamata le avrà restituito il controllo

Convenzione nell'uso e salvataggio dei registri

- Necessità di stabilire delle convenzioni
  - I registri `x10-x17 (a0-a7), x5-x7` e `x28-x31 (t0-t6)`
    - possono essere modificati dal chiamato senza nessun meccanismo di ripristino
    - Il chiamante se necessario dovrà salvare i valori dei registri prima dell'invocazione della procedura
  - I registri `x1(ra), x2(sp), x3(gp), x4 (tp), x8(fp), x9` e `x18-x27 (s1-s11)`
    - Se modificati dal chiamato devono essere salvati e poi ripristinati prima del ritorno al chiamante
    - Il chiamante non è tenuto al loro salvataggio e ripristino

Le fasi di una invocazione di procedura

Possiamo dividere l'invocazione di una procedura in 7 fasi:

1. Pre-chiamata
2. Invocazione della procedura

Chiamante

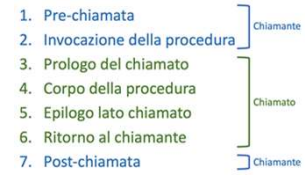
3. Prologo del chiamato
4. Corpo della procedura
5. Epilogo lato chiamato
6. Ritorno al chiamante

Chiamato

7. Post-chiamata

Chiamante

Le fasi di una invocazione di procedura



- Fase 1 – Pre-chiamata del chiamante
  - Eventuale salvataggio registri da preservare nel chiamante
    - Si assume che `x10-x17 (a0-a7)`, `x5-x7` e `x28-x31 (t0-t6)`, possano essere sovrascritti dal chiamato
      - se li si vuole preservare vanno salvati nello stack (dal chiamante) – vedi caso 1
      - il caso 2 mostra un caso in cui non è necessario salvare il contenuto del registro associato alla variabile `f`
  - Preparazione degli argomenti della funzione
    - I primi 8 argomenti vengono posti in `x10-x17`, ovvero `a0-a7` (nuovi valori)
    - Gli eventuali altri argomenti oltre l'ottavo vanno salvati nello stack (EXTRA_ARGS), così che si trovino subito sopra il frame della funzione chiamata

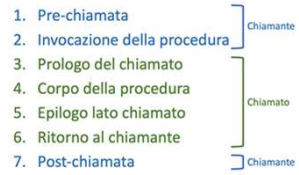
```
int somma(int x, int y) {
  x=x+y;
  return x;
}
….
f=f+1;
risultato=somma(f,g);
printf("%d", f);
return;
```

1

```
int somma(int x, int y) {
  x=x+y;
  return x;
}
….
f=f+1;
risultato=somma(f,g);
return;
```

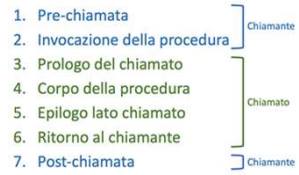
2

Le fasi di una invocazione di procedura



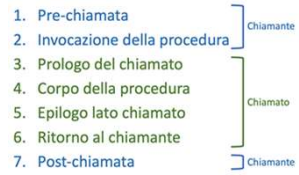
- Fase 2 – Invocazione della procedura
  - Istruzione `jal NOME_PROCEDURA`
- Fase 3 – Prologo lato chiamato
  - Eventuale allocazione del call-frame sullo stack (aggiornare `sp`)
    - Eventuale salvataggio registri che si intende sovrascrivere
      - Salvataggio degli argomenti `x10-x17` (`a0-a7`) solo se la funzione ha necessità di riusarli nel corpo di questa funzione, successivamente a ulteriori chiamate a funzione che usino tali registri, (nota: negli altri casi `x10-x17` possono essere sovrascritti)
    - Salvataggio di `x1(ra)` nel caso in cui la procedura non sia foglia
    - Salvataggio di `x8(fp)`, solo se utilizzato all'interno della procedura
    - Salvataggio di `x9 e x18-x27 (s1-s11)` se utilizzati all'interno della procedura (il chiamante si aspetta di trovarli intatti)
  - Eventuale inizializzazione di `fp` : punta al nuovo call-frame

Le fasi di una invocazione di procedura



- Fase 4 – Corpo della procedura
  - Istruzioni che implementano il corpo della procedura
- Fase 5 – Epilogo lato chiamato
  - Se deve essere restituito un valore dalla funzione
    - Tale valore viene posto in `x10` (e `x11`) ovvero `a0-a1`
  - I registri (se salvati) devono essere ripristinati
    - `x10-x17,` cioè `a0-a7` (nel caso siano stati salvati all'interno della funzione)
    - `x9` e `x18-x27 (s1-s11)`
    - `x1(ra)`
    - `x8(fp)`
  - Notare che `sp` deve solo essere aumentato di opportuno offset (lo stesso sottratto nella Fase 3

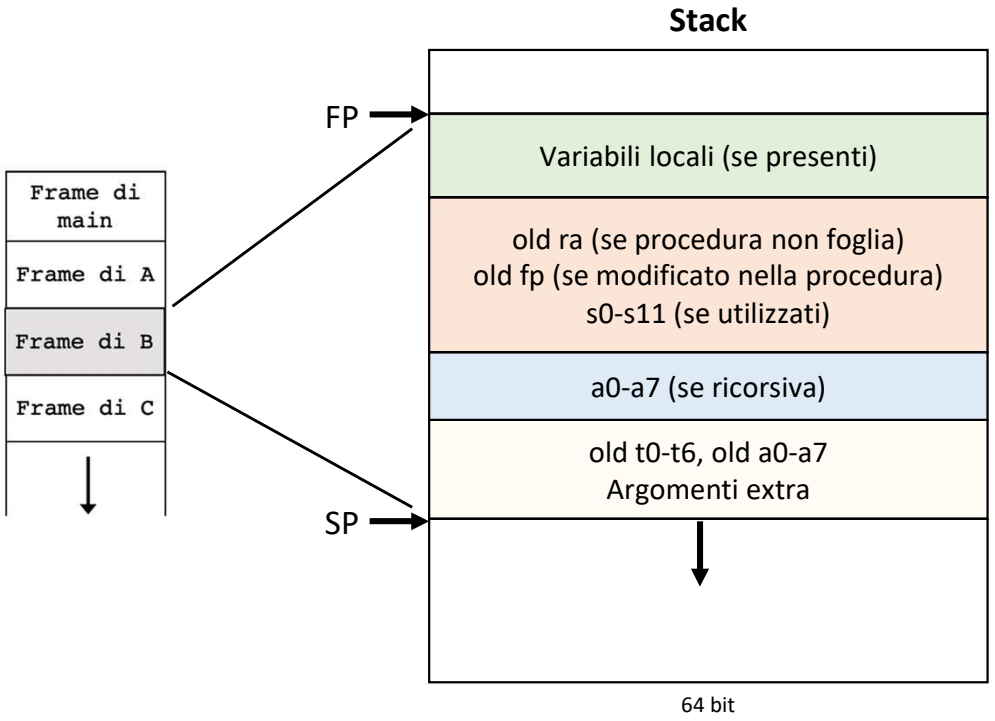
Le fasi di una invocazione di procedura



- Fase 6 – Ritorno al chiamante
  - Istruzione `jalr x0, 0(x1)` (o pseudo-istruzione `jr ra`)
- Fase 7 – Post-chiamata lato chiamante
  - Eventuale uso del risultato della funzione (in `x10` e `x11`, cioè `a0-a1`)
  - Ripristino dei valori `x5-x7` e `x28-x31` (`t0-t6`), `x10-x17` (`a0-a7`) vecchi, eventualmente salvati

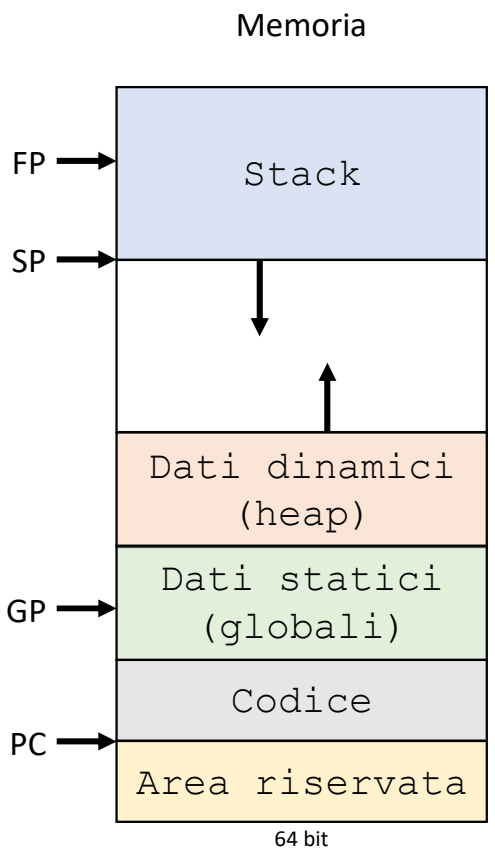
Record di attivazione: struttura

- Se utilizzato, il registro `fp` (frame pointer) viene inizializzato al valore di `sp` all'inizio della chiamata
- `fp` consente di avere un riferimento alle variabili locali che non muta con l'esecuzione della procedura
- Se lo stack non contiene variabili locali alla procedura, il compilatore risparmia tempo di esecuzione evitando di impostare e ripristinare il frame.



Organizzazione della memoria

- In memoria, oltre allo stack, trovano posto
  - Dati allocati dinamicamente (ad esempio, attraverso la malloc() del C)
  - Dati statici e variabili globali
  - Codice del programma
- Lo stack e i dati dinamici crescono in direzioni differenti, in modo da ottimizzarne la gestione



Esempio: area di un triangolo

```
long moltiplicazione(long a, long b) {
  long rst=a;
  for(long i=1;i<b;i++)
    rst = rst+a;
  return rst;
}

long area(long base, long altezza) {
  long rst = moltiplicazione(base, altezza)/2;
  return rst;
}
…
printf("Area = %li\n",area(20,23));
…
```

Si supponga di non avere una operazione di moltiplicazione

Invocazione della funzione `area` con parametri `base=20 ,altezza=23`

```
_start:
  li a0, 20 # salvo la base in a0
  li a1, 23 # salvo l'altezza in a1

# chiama area(base,altezza)
  jal  ra, area       # altezza in a0, base in a1
  add  t0, a0, zero   # salva il risultato in t0

# stampa messaggio per il risultato
  la   a0, visris   # pseudo-istruzione
  addi a7, zero,  4
  ecall

# stampa il risultato
  add  a0, t0, zero
  addi a7, zero,  1
  ecall

# stampa \n
  la   a0, RTN       # pseudo-istruzione
  addi a7, zero,  4
  ecall

# exit
  addi a7, zero,  10
  ecall
```

Il codice può essere ottimizzato

Esempio: area di un triangolo

```
long moltiplicazione(long a, long b) {
  long rst=a;
  for(long i=1;i<b;i++)
    rst = rst+a;
  return rst;
}

long area(long base, long altezza) {
  long rst = moltiplicazione(base, altezza)/2;
  return rst;
}
…
printf("Area = %li\n",area(20,23));
…
```

Si supponga di non avere una operazione di moltiplicazione

Funzione `area` con invocazione della funzione `moltiplicazione`

```
area:
# crea il call frame sullo stack (24 byte=ra+fp+rst)
# lo stack cresce verso il basso
    addi sp, sp, -24     # allocazione del call frame nello stack
    sd   ra, 8(sp)       # salvataggio dell'indirizzo di ritorno
    sd   fp, 0(sp)       # salvataggio del precedente frame pointer
    addi  fp, sp, 24     # aggiornamento del frame pointer

# calcolo dell'area
    jal ra, moltiplicazione
    sd a0, 0(fp)         # salva il risultato della moltiplicazione
                         # nella variabile locale rst
    srai a0,a0,1

# uscita dalla funzione
    ld   fp, 0(sp)       # recupera il frame pointer
    ld   ra, 8(sp)       # recupera l'indirizzo di ritorno
    addi sp, sp, 24      # elimina il call frame dallo stack
    jr   ra              # ritorna al chiamante, pseudo-istruzione
```

Il codice può essere ottimizzato

Esempio: area di un triangolo

```
long moltiplicazione(long a, long b) {
  long rst=a;
  for(long i=1;i<b;i++)
    rst = rst+a;
  return rst;
}

long area(long base, long altezza) {
  long rst = moltiplicazione(base, altezza)/2;
  return rst;
}
…
printf("Area = %li\n",area(20,23));
…
```

Si supponga di non avere una operazione di moltiplicazione

Funzione `moltiplicazione`

```
moltiplicazione:
  addi sp, sp, -16    # allocazione del call frame nello stack
  sd   ra, 8(sp)      # salvataggio dell'indirizzo di ritorno
  sd   fp, 0(sp)      # salvataggio del precedente frame pointer
  addi  fp, sp, 8     # aggiornamento del frame pointer
  sd a0, 0(fp)        # salvataggio variabile locale

  ld t2, 0(fp)
  li t0, 1
for:
  bge t0,a1,endfor
  add t2,a0,t2
  addi t0,t0,1
  j for                 # pseudo-istruzione
endfor:
  ld   fp, 0(sp)        # recupera il frame pointer
  ld   ra, 8(sp)        # recupera l'indirizzo di ritorno
  addi sp, sp, 16       # elimina il call frame dallo stack
  add a0,t2,zero        # valore di ritorno in a0
  jr   ra               # ritorna al chiamante, pseudo-istruzione
```

Il codice può essere ottimizzato

Esempio: area di un triangolo

```
long moltiplicazione(long a, long b) {
  long rst=a;
  for(long i=1;i<b;i++)
    rst = rst+a;
  return rst;
}

long area(long base, long altezza) {
  long rst = moltiplicazione(base, altezza)/2;
  return rst;
}
…
printf("Area = %li\n",area(20,23));
…
```

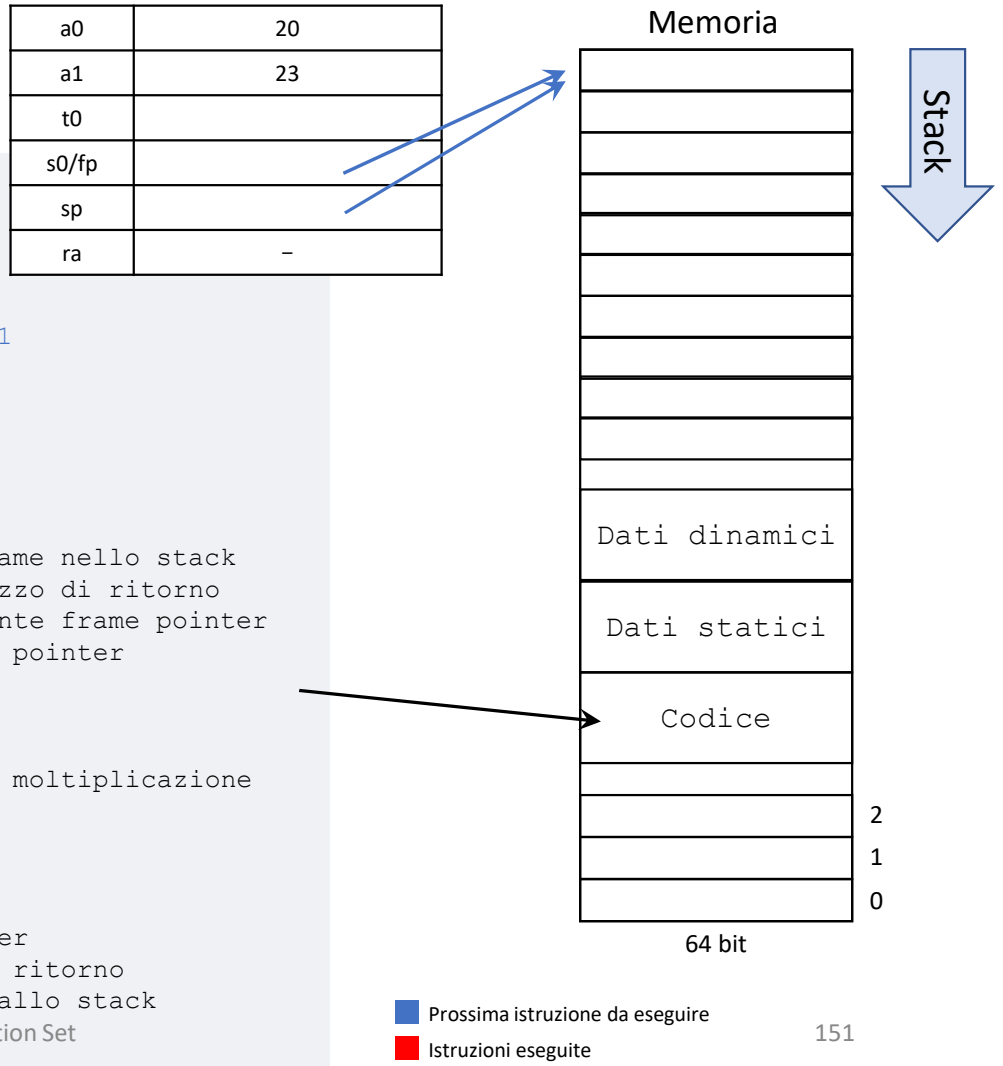
Si supponga di non avere una operazione di moltiplicazione

Il codice può essere ottimizzato

```
moltiplicazione:
  addi sp, sp, -16   # allocazione del call frame nello stack
  sd   ra, 8(sp)     # salvataggio dell'indirizzo di ritorno
  sd   fp, 0(sp)     # salvataggio del precedente frame pointer
  addi  fp, sp, 8    # aggiornamento del frame pointer
  sd a0, 0(fp)       # salvataggio variabile locale
  ld t2, 0(fp)
  li t0, 1
for:
  bge t0,a1,endfor
  add t2,a0,t2
  addi t0,t0,1
  j for                # pseudo-istruzione
endfor:
  ld   fp, 0(sp)       # recupera il frame pointer
  ld   ra, 8(sp)       # recupera l'indirizzo di ritorno
  addi sp, sp, 16      # elimina il call frame dallo stack
  add a0,t2,zero       # valore di ritorno in a0
  jr   ra              # ritorna al chiamante, pseudo-istruzione
area:
# crea il call frame sullo stack (24 byte=ra+fp+rst)
addi sp, sp, -24     # allocazione del call frame nello stack
    sd   ra, 8(sp)       # salvataggio dell'indirizzo di ritorno
    sd   fp, 0(sp)       # salvataggio del precedente frame pointer
    addi  fp, sp, 24     # aggiornamento del frame pointer
# calcolo dell'area
    jal ra, moltiplicazione
    sd a0, 0(fp)  # salva il risultato della moltiplicazione in rst
srai a0,a0,1
# uscita dalla funzione
    ld   fp, 0(sp)       # recupera il frame pointer
    ld   ra, 8(sp)       # recupera l'indirizzo di ritorno
    addi sp, sp, 24      # elimina il call frame dallo stack
    jr   ra              # ritorna al chiamante, pseudo-istruzione
_start:
  li a0, 20 # salvo la base in a0
  li a1, 23 # salvo l'altezza in a1
# chiama area(base,altezza)
  jal  ra, area       # altezza in a0, base in a1
  add  t0, a0, zero   # salva il risultato in t0
# stampa messaggio per il risultato
  la   a0, visris             # pseudo-istruzione
  addi a7, zero,  4
  ecall
# stampa il risultato
  add  a0, t0, zero
  addi a7, zero,  1
  ecall
# stampa \n
  la   a0, RTN  # pseudo-istruzione
  addi a7, zero,  4
  ecall
# exit
  addi a7, zero,  10
  ecall
```

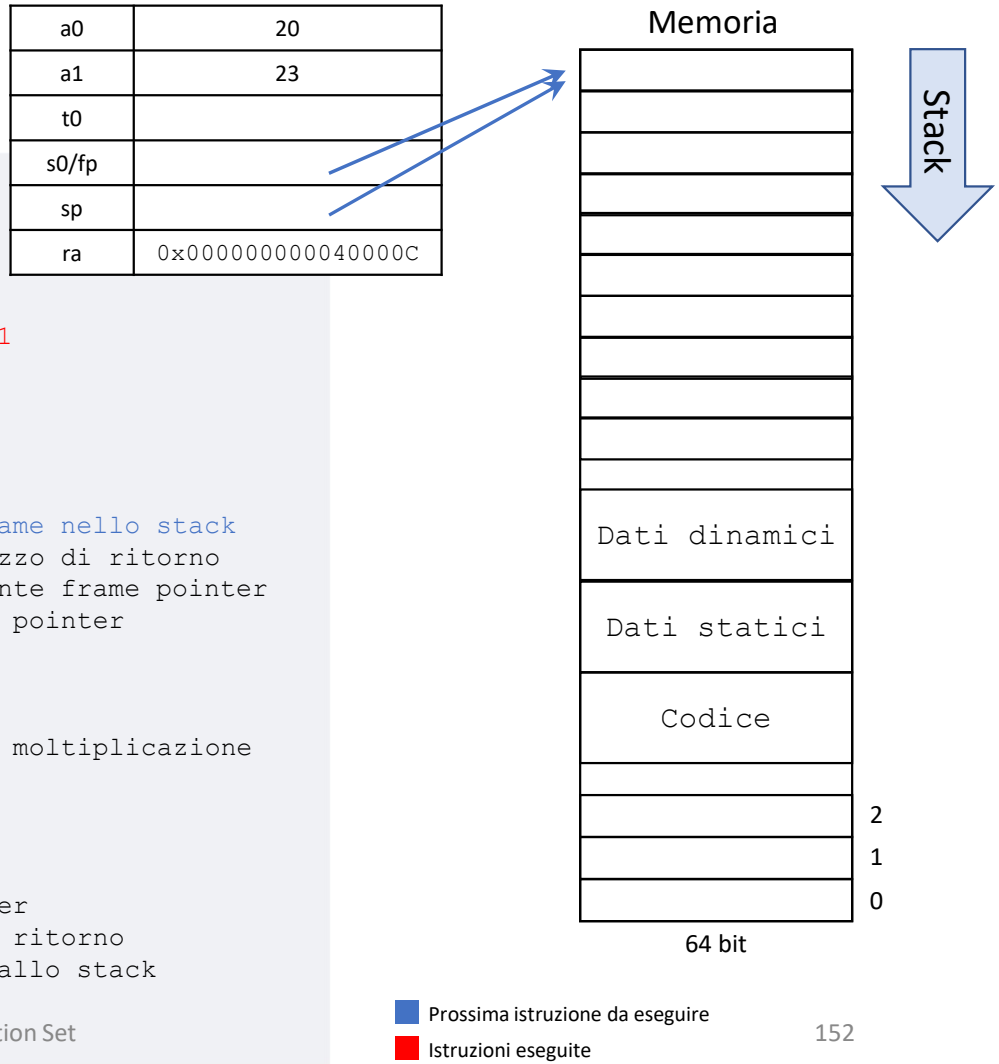
Un esempio

```
_start:
0x0000000000400000  li a0, 20 # salvo la base in a0
0x0000000000400004  li a1, 23 # salvo l'altezza in a1

# chiama triangolo(base,altezza)
0x0000000000400008  jal  ra, area       # altezza in a0, base in a1
0x000000000040000C  add  t0, a0, zero   # salva il risultato in t0
…

area:
# crea il call frame sullo stack (24 byte=ra+fp+rst)
# lo stack cresce verso il basso
0x0000000000400010  addi sp, sp, -24     # allocazione del call frame nello stack
0x0000000000400014  sd   ra, 8(sp)       # salvataggio dell'indirizzo di ritorno
0x0000000000400018  sd   fp, 0(sp)       # salvataggio del precedente frame pointer
0x000000000040001C  addi  fp, sp, 16     # aggiornamento del frame pointer

# calcolo dell'area
0x0000000000400020  jal ra, moltiplicazione
0x0000000000400024  sd a0, 0(fp)         #salva il risultato della moltiplicazione
                        #nella variabile locale rst
0x0000000000400028  srai a0,a0,1

# uscita dalla funzione
0x000000000040002C  ld   fp, 0(sp)       # recupera il frame pointer
0x0000000000400030  ld   ra, 8(sp)       # recupera l'indirizzo di ritorno
0x0000000000400034  addi sp, sp, 24      # elimina il call frame dallo stack
0x0000000000400038  jr   ra              # ritorna al chiamante
…
```



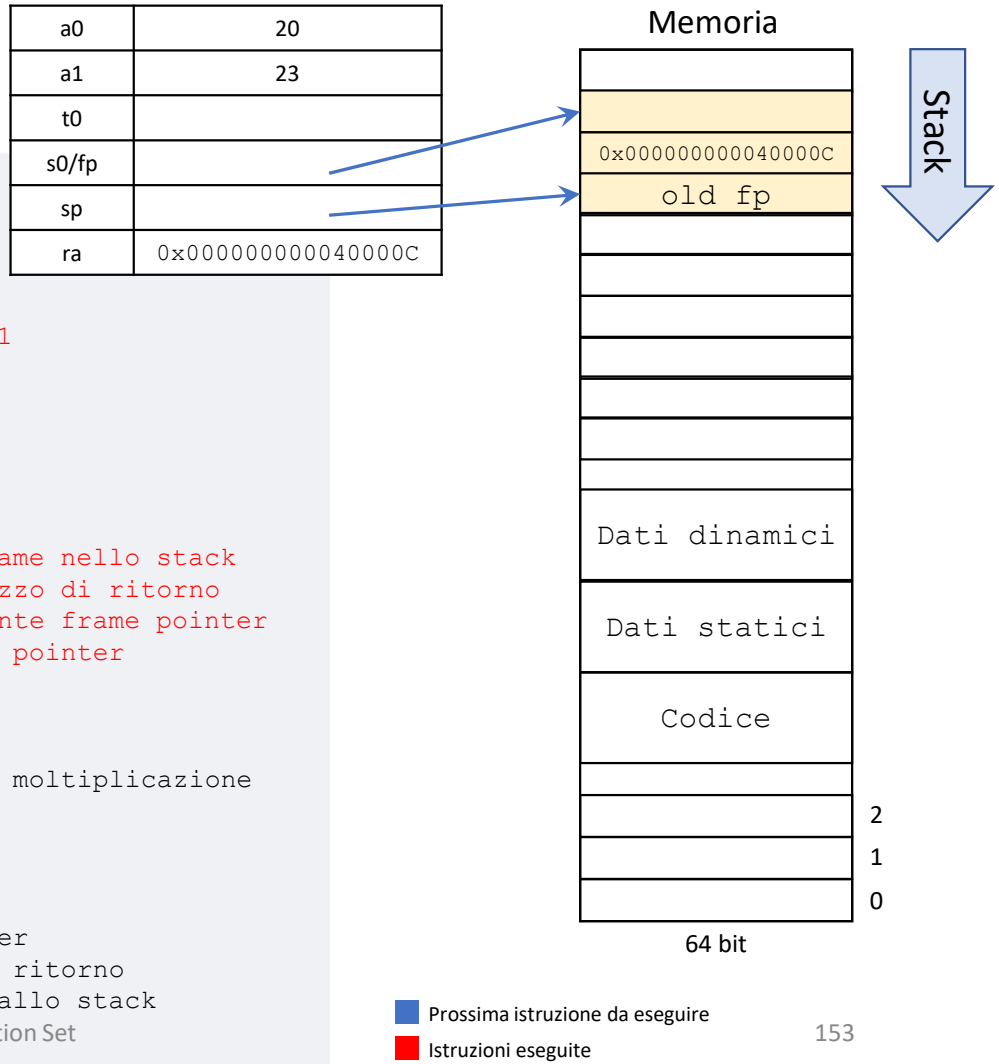
Un esempio

```
_start:
0x0000000000400000  li a0, 20 # salvo la base in a0
0x0000000000400004  li a1, 23 # salvo l'altezza in a1

# chiama triangolo(base,altezza)
0x0000000000400008  jal  ra, area       # altezza in a0, base in a1
0x000000000040000C  add  t0, a0, zero   # salva il risultato in t0
…
area:
# crea il call frame sullo stack (24 byte=ra+fp+rst)
# lo stack cresce verso il basso
0x0000000000400010  addi sp, sp, -24     # allocazione del call frame nello stack
0x0000000000400014  sd   ra, 8(sp)       # salvataggio dell'indirizzo di ritorno
0x0000000000400018  sd   fp, 0(sp)       # salvataggio del precedente frame pointer
0x000000000040001C  addi  fp, sp, 16     # aggiornamento del frame pointer

# calcolo dell'area
0x0000000000400020  jal ra, moltiplicazione
0x0000000000400024  sd a0, 0(fp)         #salva il risultato della moltiplicazione
                        #nella variabile locale rst
0x0000000000400028  srai a0,a0,1

# uscita dalla funzione
0x000000000040002C  ld   fp, 0(sp)       # recupera il frame pointer
0x0000000000400030  ld   ra, 8(sp)       # recupera l'indirizzo di ritorno
0x0000000000400034  addi sp, sp, 24      # elimina il call frame dallo stack
0x0000000000400038  jr   ra              # ritorna al chiamante
…
```



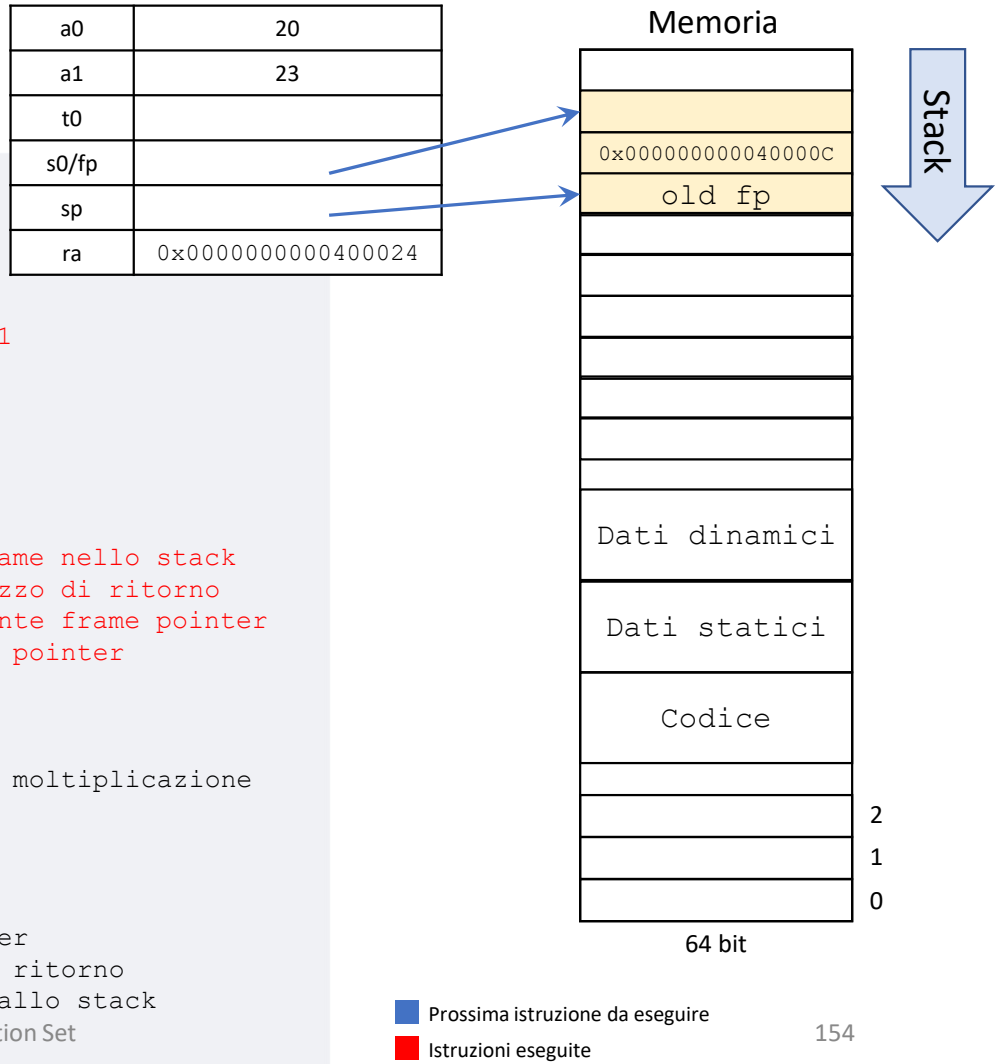
Un esempio

```
_start:
0x0000000000400000  li a0, 20 # salvo la base in a0
0x0000000000400004  li a1, 23 # salvo l'altezza in a1

# chiama triangolo(base,altezza)
0x0000000000400008  jal  ra, area       # altezza in a0, base in a1
0x000000000040000C  add  t0, a0, zero   # salva il risultato in t0
…

area:
# crea il call frame sullo stack (24 byte=ra+fp+rst)
# lo stack cresce verso il basso
0x0000000000400010  addi sp, sp, -24     # allocazione del call frame nello stack
0x0000000000400014  sd   ra, 8(sp)       # salvataggio dell'indirizzo di ritorno
0x0000000000400018  sd   fp, 0(sp)       # salvataggio del precedente frame pointer
0x000000000040001C  addi  fp, sp, 16     # aggiornamento del frame pointer

# calcolo dell'area
0x0000000000400020  jal ra, moltiplicazione
0x0000000000400024  sd a0, 0(fp)         #salva il risultato della moltiplicazione
                        #nella variabile locale rst
0x0000000000400028  srai a0,a0,1

# uscita dalla funzione
0x000000000040002C  ld   fp, 0(sp)       # recupera il frame pointer
0x0000000000400030  ld   ra, 8(sp)       # recupera l'indirizzo di ritorno
0x0000000000400034  addi sp, sp, 24      # elimina il call frame dallo stack
0x0000000000400038  jr   ra              # ritorna al chiamante
…
```



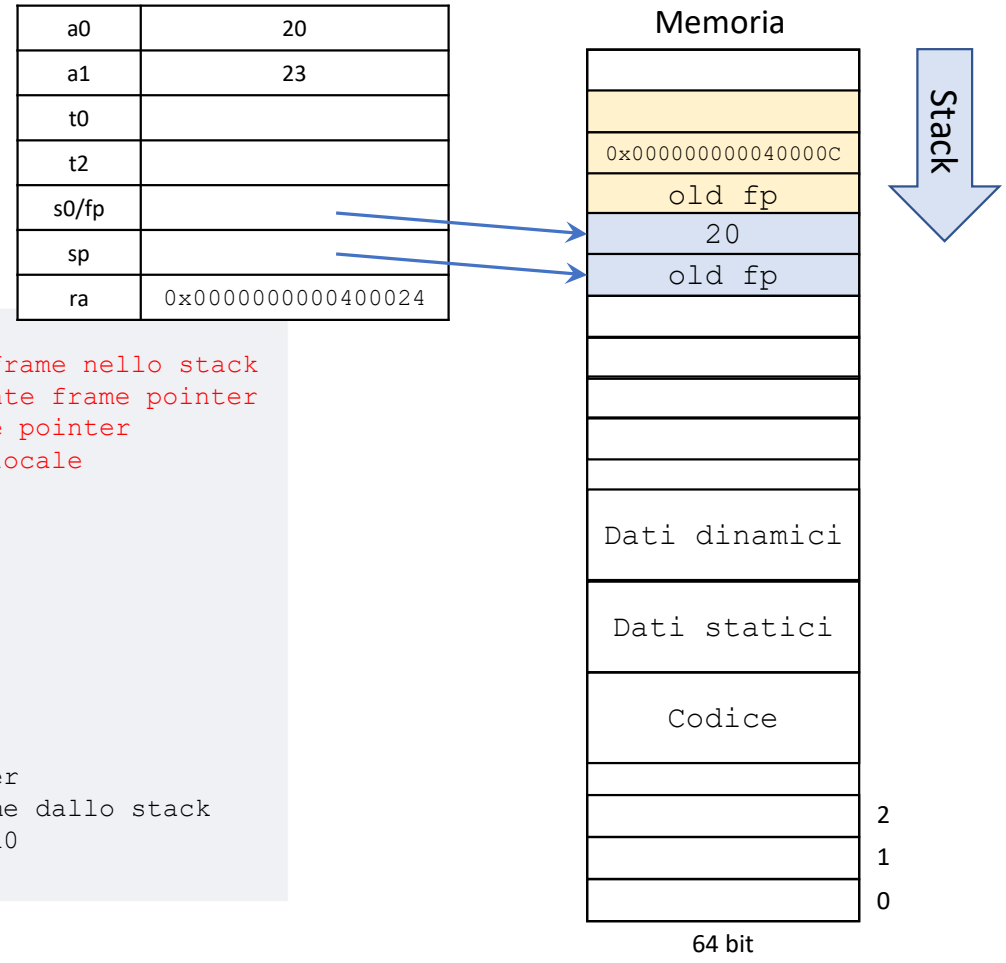
Un esempio

```
_start:
0x0000000000400000  li a0, 20 # salvo la base in a0
0x0000000000400004  li a1, 23 # salvo l'altezza in a1

# chiama triangolo(base,altezza)
0x0000000000400008  jal  ra, area       # altezza in a0, base in a1
0x000000000040000C  add  t0, a0, zero   # salva il risultato in t0
…

area:
# crea il call frame sullo stack (24 byte=ra+fp+rst)
# lo stack cresce verso il basso
0x0000000000400010  addi sp, sp, -24     # allocazione del call frame nello stack
0x0000000000400014  sd   ra, 8(sp)       # salvataggio dell'indirizzo di ritorno
0x0000000000400018  sd   fp, 0(sp)       # salvataggio del precedente frame pointer
0x000000000040001C  addi  fp, sp, 16     # aggiornamento del frame pointer

# calcolo dell'area
0x0000000000400020  jal ra, moltiplicazione
0x0000000000400024  sd a0, 0(fp)         #salva il risultato della moltiplicazione
                        #nella variabile locale rst
0x0000000000400028  srai a0,a0,1

# uscita dalla funzione
0x000000000040002C  ld   fp, 0(sp)       # recupera il frame pointer
0x0000000000400030  ld   ra, 8(sp)       # recupera l'indirizzo di ritorno
0x0000000000400034  addi sp, sp, 24      # elimina il call frame dallo stack
0x0000000000400038  jr   ra              # ritorna al chiamante
…
```



Un esempio

```
moltiplicazione:
0x0000000000400038  addi sp, sp, -16      # allocazione del call frame nello stack
0x000000000040003C  sd   fp, 0(sp)       # salvataggio del precedente frame pointer
0x0000000000400040  addi  fp, sp, 0      # aggiornamento del frame pointer
0x0000000000400044  sd a0, 0(fp)         # salvataggio variabile locale

0x0000000000400048  ld t2, 0(fp)
0x000000000040004C  li t0, 1
for:
0x0000000000400050  bge t0,a1,endfor
0x0000000000400054  add t2,a0,t2
0x0000000000400058  addi t0,t0,1
0x000000000040005C  j for
endfor:
0x0000000000400060  ld fp, 0(sp)        # recupera il frame pointer
0x0000000000400064  addi sp, sp, 16        # elimina il call frame dallo stack
0x0000000000400068  add a0,t2,zero        #valore di ritorno in a0
0x000000000040006C  jr   ra               # ritorna al chiamante
```

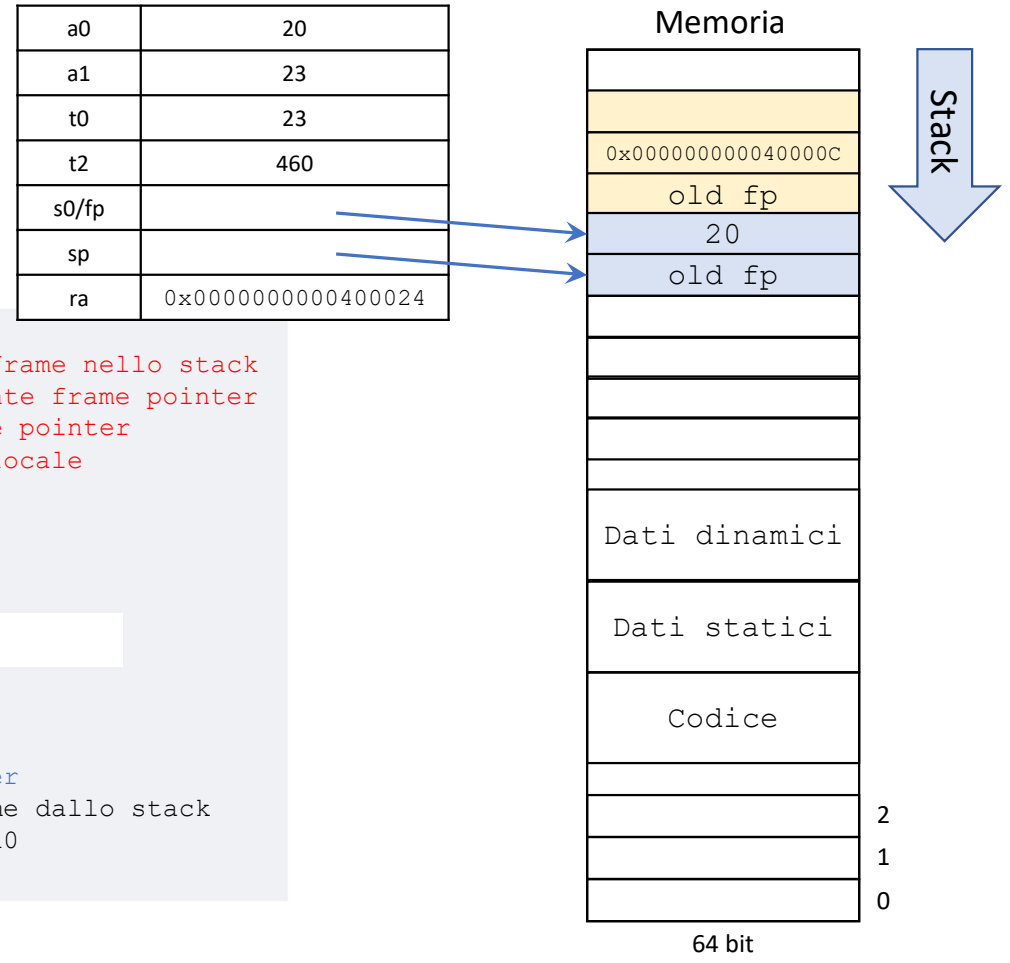


Un esempio

```
moltiplicazione:
0x0000000000400038  addi sp, sp, -16      # allocazione del call frame nello stack
0x000000000040003C  sd   fp, 0(sp)      # salvataggio del precedente frame pointer
0x0000000000400040  addi  fp, sp, 0      # aggiornamento del frame pointer
0x0000000000400044  sd a0, 0(fp)         # salvataggio variabile locale

0x0000000000400048  ld t2, 0(fp)
0x000000000040004C  li t0, 1
for:
0x0000000000400050  bge t0,a1,endfor
0x0000000000400054  add t2,a0,t2
0x0000000000400058  addi t0,t0,1
0x000000000040005C  j for
endfor:
0x0000000000400060  ld fp, 0(sp)        # recupera il frame pointer
0x0000000000400064  addi sp, sp, 16        # elimina il call frame dallo stack
0x0000000000400068  add a0,t2,zero        #valore di ritorno in a0
0x000000000040006C  jr   ra               # ritorna al chiamante
```

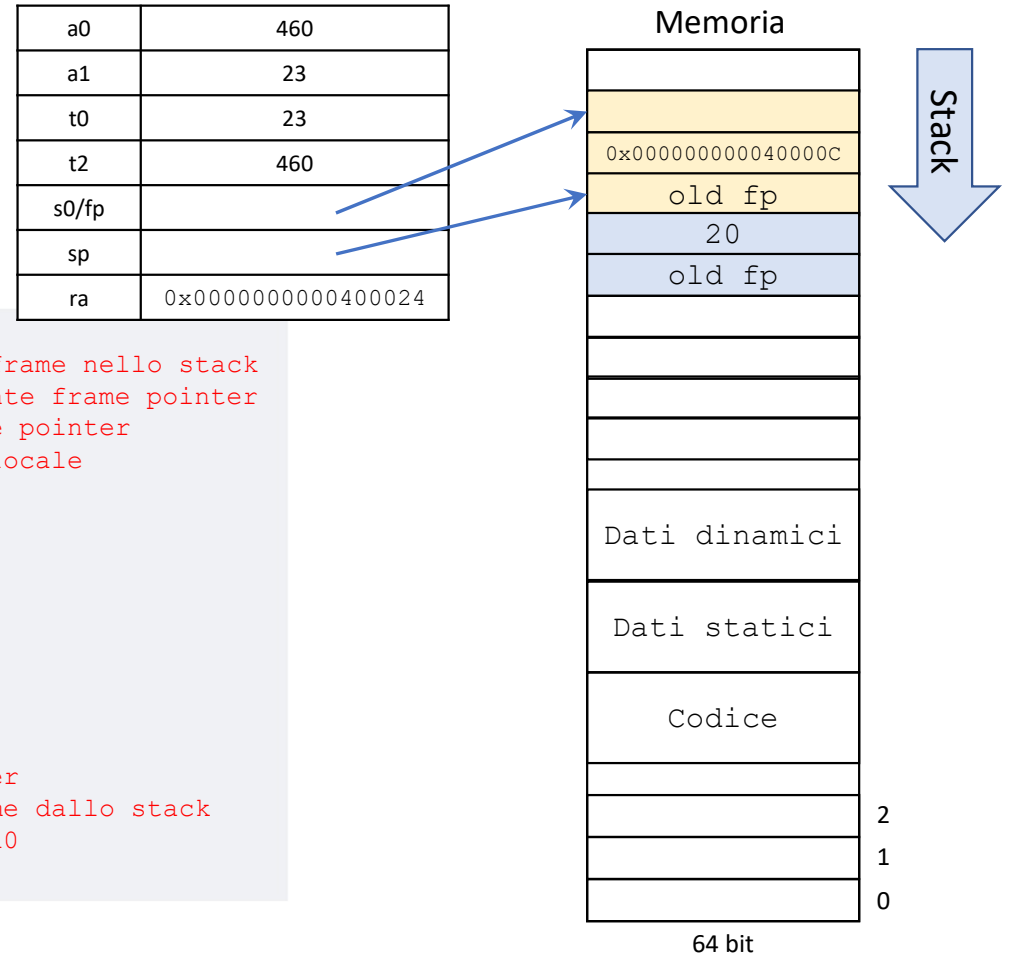
Al termine del ciclo for…

a0	20
a1	23
t0	23
t2	460
s0/fp	
sp	
ra	0x0000000000400024

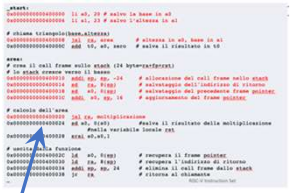


Un esempio

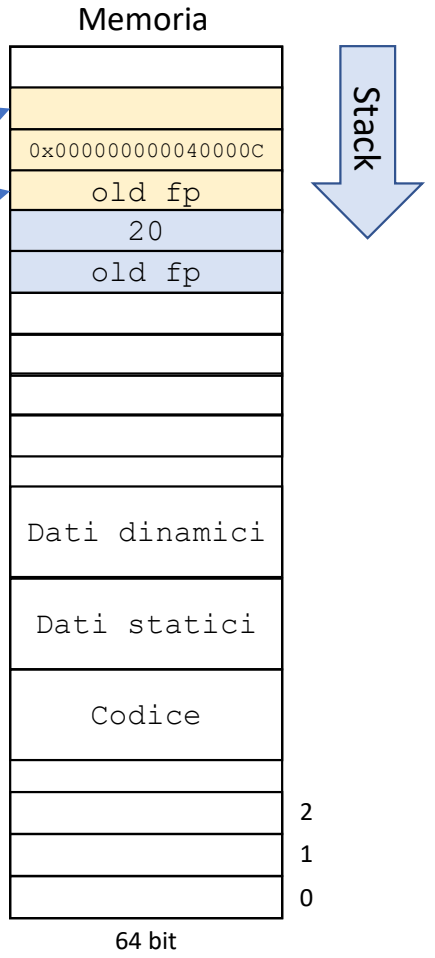
```
moltiplicazione:
0x0000000000400038  addi sp, sp, -16     # allocazione del call frame nello stack
0x000000000040003C  sd   fp, 0(sp)      # salvataggio del precedente frame pointer
0x0000000000400040  addi  fp, sp, 0      # aggiornamento del frame pointer
0x0000000000400044  sd a0, 0(fp)         # salvataggio variabile locale

0x0000000000400048  ld t2, 0(fp)
0x000000000040004C  li t0, 1
for:
0x0000000000400050  bge t0,a1,endfor
0x0000000000400054  add t2,a0,t2
0x0000000000400058  addi t0,t0,1
0x000000000040005C  j for
endfor:
0x0000000000400060  ld fp, 0(sp)        # recupera il frame pointer
0x0000000000400064  addi sp, sp, 16        # elimina il call frame dallo stack
0x0000000000400068  add a0,t2,zero        #valore di ritorno in a0
0x000000000040006C  jr   ra               # ritorna al chiamante
```



Un esempio



a0	460
a1	23
t0	23
t2	460
s0/fp	
sp	
ra	0x0000000000400024




```
moltiplicazione:
0x0000000000400038  addi sp, sp, -16      # allocazione del call frame nello stack
0x000000000040003C  sd   fp, 0(sp)      # salvataggio del precedente frame pointer
0x0000000000400040  addi  fp, sp, 0      # aggiornamento del frame pointer
0x0000000000400044  sd a0, 0(fp)         # salvataggio variabile locale

0x0000000000400048  ld t2, 0(fp)
0x000000000040004C  li t0, 1
for:
0x0000000000400050  bge t0,a1,endfor
0x0000000000400054  add t2,a0,t2
0x0000000000400058  addi t0,t0,1
0x000000000040005C  j for
endfor:
0x0000000000400060  ld fp, 0(sp)        # recupera il frame pointer
0x0000000000400064  addi sp, sp, 16        # elimina il call frame dallo stack
0x0000000000400068  add a0,t2,zero        #valore di ritorno in a0
0x000000000040006C  jr   ra               # ritorna al chiamante
```

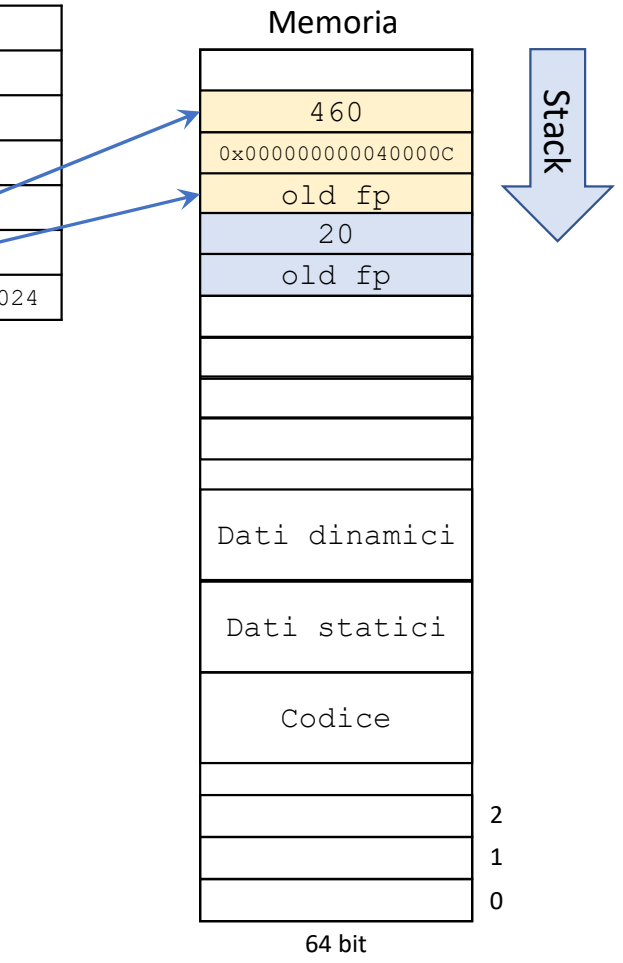
Prossima istruzione da eseguire

Istruzioni eseguite

Un esempio

```
_start:
0x0000000000400000  li a0, 20 # salvo la base in a0
0x0000000000400004  li a1, 23 # salvo l'altezza in a1

# chiama triangolo(base,altezza)
0x0000000000400008  jal  ra, area       # altezza in a0, base in a1
0x000000000040000C  add  t0, a0, zero   # salva il risultato in t0
…

area:
# crea il call frame sullo stack (24 byte=ra+fp+rst)
# lo stack cresce verso il basso
0x0000000000400010  addi sp, sp, -24     # allocazione del call frame nello stack
0x0000000000400014  sd   ra, 8(sp)       # salvataggio dell'indirizzo di ritorno
0x0000000000400018  sd   fp, 0(sp)       # salvataggio del precedente frame pointer
0x000000000040001C  addi  fp, sp, 16     # aggiornamento del frame pointer

# calcolo dell'area
0x0000000000400020  jal ra, moltiplicazione
0x0000000000400024  sd a0, 0(fp)         #salva il risultato della moltiplicazione
                        #nella variabile locale rst
0x0000000000400028  srai a0,a0,1.        #divide per due

# uscita dalla funzione
0x000000000040002C  ld   fp, 0(sp)       # recupera il frame pointer
0x0000000000400030  ld   ra, 8(sp)       # recupera l'indirizzo di ritorno
0x0000000000400034  addi sp, sp, 24      # elimina il call frame dallo stack
0x0000000000400038  jr   ra              # ritorna al chiamante
…
```

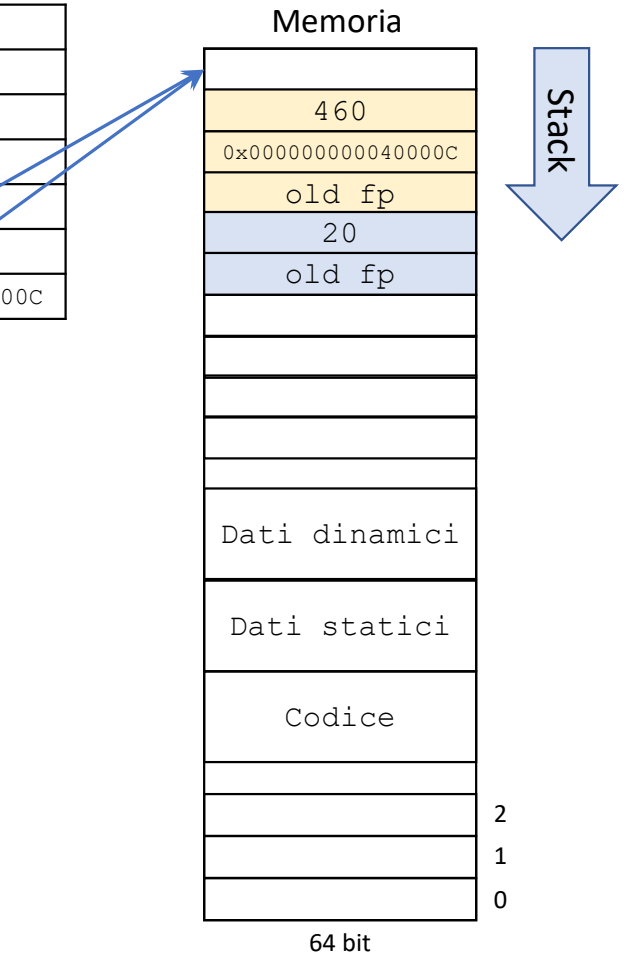
a0	230
a1	23
t0	23
t2	460
s0/fp	
sp	
ra	0x0000000000400024



Un esempio

```
_start:
0x0000000000400000  li a0, 20 # salvo la base in a0
0x0000000000400004  li a1, 23 # salvo l'altezza in a1

# chiama triangolo(base,altezza)
0x0000000000400008  jal  ra, area       # altezza in a0, base in a1
0x000000000040000C  add  t0, a0, zero   # salva il risultato in t0
…

area:
# crea il call frame sullo stack (24 byte=ra+fp+rst)
# lo stack cresce verso il basso
0x0000000000400010  addi sp, sp, -24     # allocazione del call frame nello stack
0x0000000000400014  sd   ra, 8(sp)       # salvataggio dell'indirizzo di ritorno
0x0000000000400018  sd   fp, 0(sp)       # salvataggio del precedente frame pointer
0x000000000040001C  addi  fp, sp, 16     # aggiornamento del frame pointer

# calcolo dell'area
0x0000000000400020  jal ra, moltiplicazione
0x0000000000400024  sd a0, 0(fp)         #salva il risultato della moltiplicazione
                        #nella variabile locale rst
0x0000000000400028  srai a0,a0,1.        #divide per due

# uscita dalla funzione
0x000000000040002C  ld   fp, 0(sp)       # recupera il frame pointer
0x0000000000400030  ld   ra, 8(sp)       # recupera l'indirizzo di ritorno
0x0000000000400034  addi sp, sp, 24      # elimina il call frame dallo stack
0x0000000000400038  jr   ra              # ritorna al chiamante
…
```

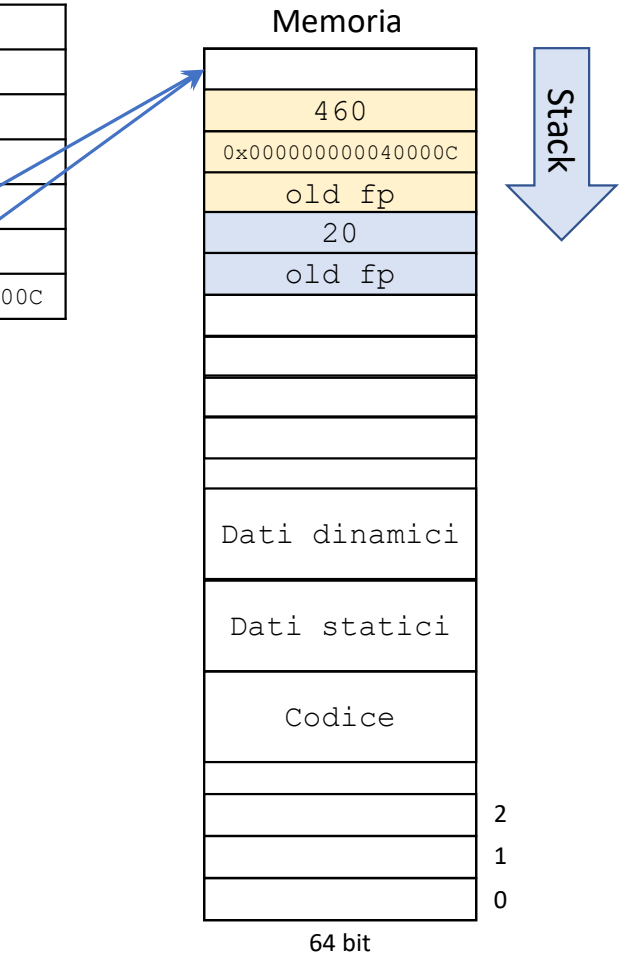
a0	230
a1	23
t0	23
t2	460
s0/fp	
sp	
ra	0x000000000040000C



Un esempio

```
_start:
0x0000000000400000  li a0, 20 # salvo la base in a0
0x0000000000400004  li a1, 23 # salvo l'altezza in a1

# chiama triangolo(base,altezza)
0x0000000000400008  jal  ra, area       # altezza in a0, base in a1
0x000000000040000C  add  t0, a0, zero   # salva il risultato in t0
…

area:
# crea il call frame sullo stack (24 byte=ra+fp+rst)
# lo stack cresce verso il basso
0x0000000000400010  addi sp, sp, -24     # allocazione del call frame nello stack
0x0000000000400014  sd   ra, 8(sp)       # salvataggio dell'indirizzo di ritorno
0x0000000000400018  sd   fp, 0(sp)       # salvataggio del precedente frame pointer
0x000000000040001C  addi  fp, sp, 16     # aggiornamento del frame pointer

# calcolo dell'area
0x0000000000400020  jal ra, moltiplicazione
0x0000000000400024  sd a0, 0(fp)         #salva il risultato della moltiplicazione
                        #nella variabile locale rst
0x0000000000400028  srai a0,a0,1.        #divide per due

# uscita dalla funzione
0x000000000040002C  ld   fp, 0(sp)       # recupera il frame pointer
0x0000000000400030  ld   ra, 8(sp)       # recupera l'indirizzo di ritorno
0x0000000000400034  addi sp, sp, 24      # elimina il call frame dallo stack
0x0000000000400038  jr   ra              # ritorna al chiamante
…
```

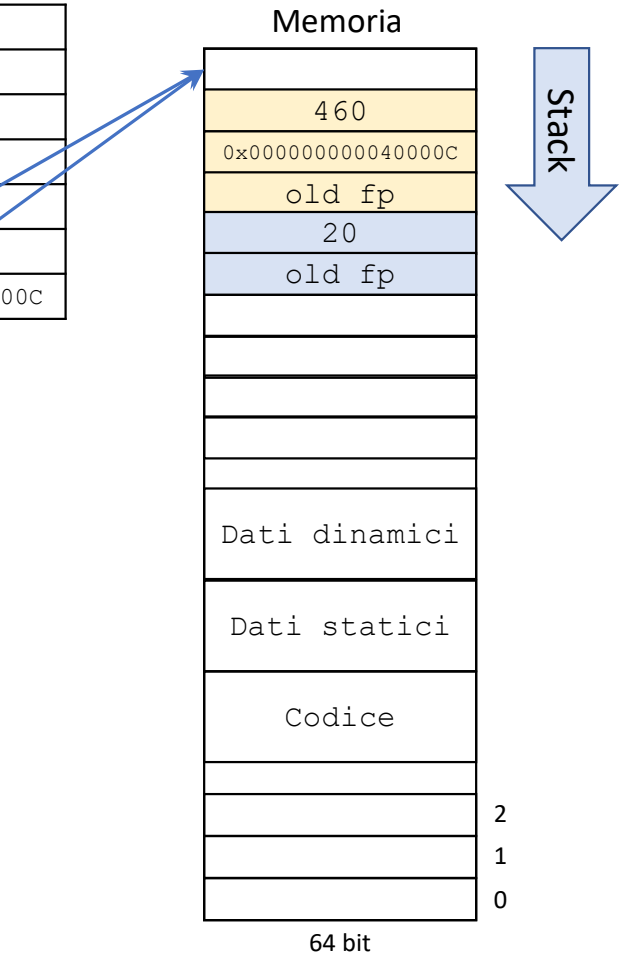
a0	230
a1	23
t0	23
t2	460
s0/fp	
sp	
ra	0x000000000040000C



Un esempio

```
_start:
0x0000000000400000  li a0, 20 # salvo la base in a0
0x0000000000400004  li a1, 23 # salvo l'altezza in a1

# chiama triangolo(base,altezza)
0x0000000000400008  jal  ra, area       # altezza in a0, base in a1
0x000000000040000C  add  t0, a0, zero   # salva il risultato in t0
…

area:
# crea il call frame sullo stack (24 byte=ra+fp+rst)
# lo stack cresce verso il basso
0x0000000000400010  addi sp, sp, -24     # allocazione del call frame nello stack
0x0000000000400014  sd   ra, 8(sp)       # salvataggio dell'indirizzo di ritorno
0x0000000000400018  sd   fp, 0(sp)       # salvataggio del precedente frame pointer
0x000000000040001C  addi  fp, sp, 16     # aggiornamento del frame pointer

# calcolo dell'area
0x0000000000400020  jal ra, moltiplicazione
0x0000000000400024  sd a0, 0(fp)         #salva il risultato della moltiplicazione
                        #nella variabile locale rst
0x0000000000400028  srai a0,a0,1.        #divide per due

# uscita dalla funzione
0x000000000040002C  ld   fp, 0(sp)       # recupera il frame pointer
0x0000000000400030  ld   ra, 8(sp)       # recupera l'indirizzo di ritorno
0x0000000000400034  addi sp, sp, 24      # elimina il call frame dallo stack
0x0000000000400038  jr   ra              # ritorna al chiamante
…
```

a0	230
a1	23
t0	230
t2	460
s0/fp	
sp	
ra	0x000000000040000C



Esempio: calcolo del fattoriale

```
int fact(int n) {
  if (n==0)
    return 1;
  else return n*fact(n-1)

}
```

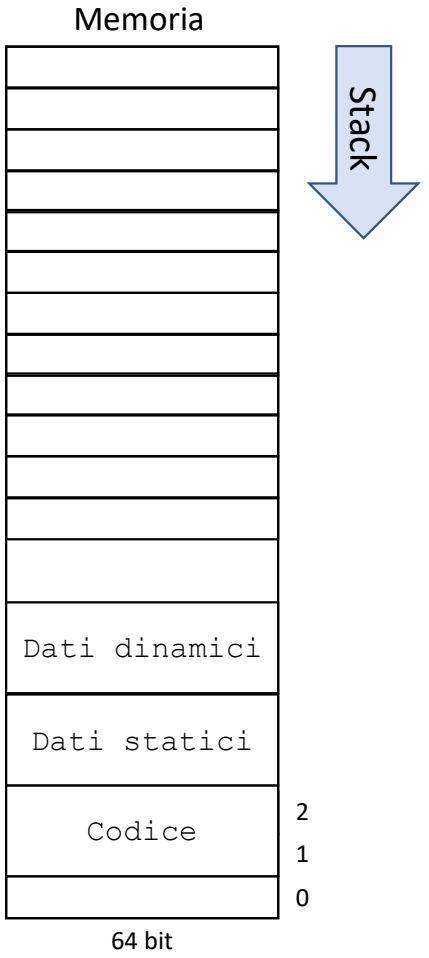
```
fact:
# crea il call frame sullo stack (24 byte)
# lo stack cresce verso il basso
    addi sp, sp, -24     # allocazione del call frame nello stack
    sd   a0, 16(sp)       # salvataggio di n nel call frame
    sd   ra, 8(sp)       # salvataggio dell'indirizzo di ritorno
    sd   fp, 0(sp)       # salvataggio del precedente frame pointer
    addi  fp, sp, 16    # aggiornamento del frame pointer

# calcolo del fattoriale
    bne  a0, zero,  Ric  # test fine ricorsione n!=0
    addi a0, zero, 1     # 0! = 1
    j    Fine

Ric:                     # chiamata ricorsiva per il calcolo di (n-1)!
    addi a0, a0, -1      # a0 <- (n – 1)  passaggio del parametro in a0 per fact(n-1)
    jal  fact            # chiama fact(n-1) -> risultato in a0
    ld   t0, 0(fp)       # t0 <- n
    mul  a0, a0, t0      # n! = (n-1)! x n

# uscita dalla funzione
Fine:
    ld   fp, 0(sp)       # recupera il frame pointer
    ld   ra, 8(sp)       # recupera l'indirizzo di ritorno
    addi sp, sp, 24      # elimina il call frame dallo stack
    jr   ra              # ritorna al chiamante
```

Esempio: calcolo del fattoriale

```
_start:
0x400000  li a0, 3 # salvo n in a0

0x400004  jal  ra, fact          # n in a0
0x400008  add  t0, a0, zero      # salva il risultato in t0

fact:
0x40000c  addi sp, sp, -24       # allocazione del call frame nello stack
0x400010  sd   a0, 16(sp)        # salvataggio di n nel call frame
0x400014  sd   ra, 8(sp)         # salvataggio dell'indirizzo di ritorno
0x400018  sd   fp, 0(sp)         # salvataggio del precedente frame pointer
0x40001c  addi  fp, sp, 16       # aggiornamento del frame pointer

0x400020  bne  a0, zero,  Ric # test fine ricorsione n!=0
0x400024  addi a0, zero, 1    # 0! = 1
0x400028  j    Fine

ric:                          # chiamata ricorsiva per il calcolo di (n-1)!
0x40002c  addi a0, a0, -1     # a0 <- (n - 1) parametro per fact(n-1)
0x400030  jal  fact           # chiama fact(n-1) -> risultato in a0
0x400034  ld   t0, 0(fp)      # t0 <- n
0x400038  mul  a0, a0, t0     # n! = (n-1)! x n

fine:
0x40003c  ld   fp, 0(sp)      # recupera il frame pointer
0x400040  ld   ra, 8(sp)      # recupera l'indirizzo di ritorno
0x400044  addi sp, sp, 24     # elimina il call frame dallo stack
0x400048  jr   ra             # ritorna al chiamante
```

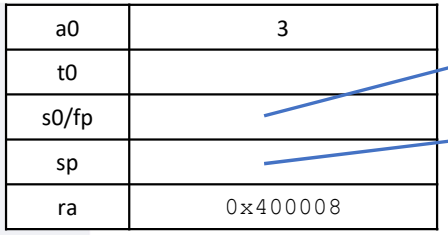
Prima dell'invocazione di fact(3)

a0	3
t0	
s0/fp	
sp	
ra	

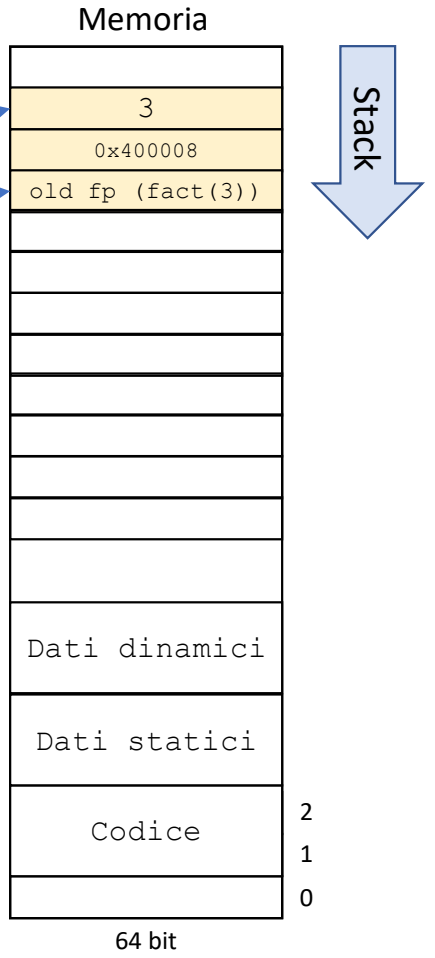


Esempio: calcolo del fattoriale

```
_start:
0x400000  li a0, 3 # salvo n in a0

0x400004  jal  ra, fact       # n in a0
0x400008  add  t0, a0, zero   # salva il risultato in t0

fact:
0x40000c  addi sp, sp, -24    # allocazione del call frame nello stack
0x400010  sd   a0, 16(sp)     # salvataggio di n nel call frame
0x400014  sd   ra, 8(sp)      # salvataggio dell'indirizzo di ritorno
0x400018  sd   fp, 0(sp)      # salvataggio del precedente frame pointer
0x40001c  addi  fp, sp, 16    # aggiornamento del frame pointer

0x400020  bne  a0, zero,  Ric # test fine ricorsione n!=0
0x400024  addi a0, zero, 1    # 0! = 1
0x400028  j    Fine

ric:                          # chiamata ricorsiva per il calcolo di (n-1)!
0x40002c  addi a0, a0, -1     # a0 <- (n - 1) parametro per fact(n-1)
0x400030  jal  fact           # chiama fact(n-1) -> risultato in a0
0x400034  ld   t0, 0(fp)      # t0 <- n
0x400038  mul  a0, a0, t0     # n! = (n-1)! x n

fine:
0x40003c  ld   fp, 0(sp)      # recupera il frame pointer
0x400040  ld   ra, 8(sp)      # recupera l'indirizzo di ritorno
0x400044  addi sp, sp, 24     # elimina il call frame dallo stack
0x400048  jr   ra             # ritorna al chiamante
```



Dopo l'invocazione e della costruzione del record di attivazione di fact(3)

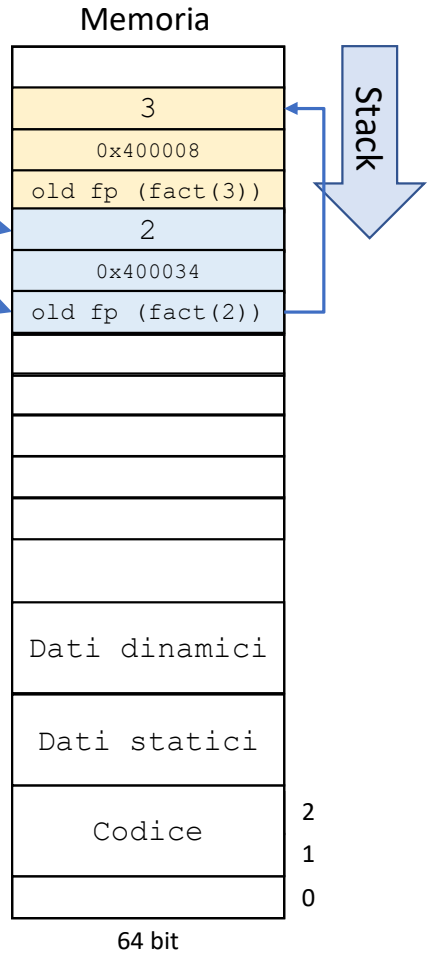


Esempio: calcolo del fattoriale

```
_start:
0x400000  li a0, 3 # salvo n in a0

0x400004  jal  ra, fact       # n in a0
0x400008  add  t0, a0, zero   # salva il risultato in t0

fact:
0x40000c  addi sp, sp, -24    # allocazione del call frame nello stack
0x400010  sd   a0, 16(sp)     # salvataggio di n nel call frame
0x400014  sd   ra, 8(sp)      # salvataggio dell'indirizzo di ritorno
0x400018  sd   fp, 0(sp)      # salvataggio del precedente frame pointer
0x40001c  addi  fp, sp, 16    # aggiornamento del frame pointer

0x400020  bne  a0, zero,  Ric # test fine ricorsione n!=0
0x400024  addi a0, zero, 1    # 0! = 1
0x400028  j    Fine

ric:                          # chiamata ricorsiva per il calcolo di (n-1)!
0x40002c  addi a0, a0, -1     # a0 <- (n - 1) parametro per fact(n-1)
0x400030  jal  fact           # chiama fact(n-1) -> risultato in a0
0x400034  ld   t0, 0(fp)      # t0 <- n
0x400038  mul  a0, a0, t0     # n! = (n-1)! x n

fine:
0x40003c  ld   fp, 0(sp)      # recupera il frame pointer
0x400040  ld   ra, 8(sp)      # recupera l'indirizzo di ritorno
0x400044  addi sp, sp, 24     # elimina il call frame dallo stack
0x400048  jr   ra             # ritorna al chiamante
```

a0	2
t0	
s0/fp	
sp	
ra	0x400034

Dopo la chiamata ricorsiva e la costruzione del record di attivazione di fact(2)

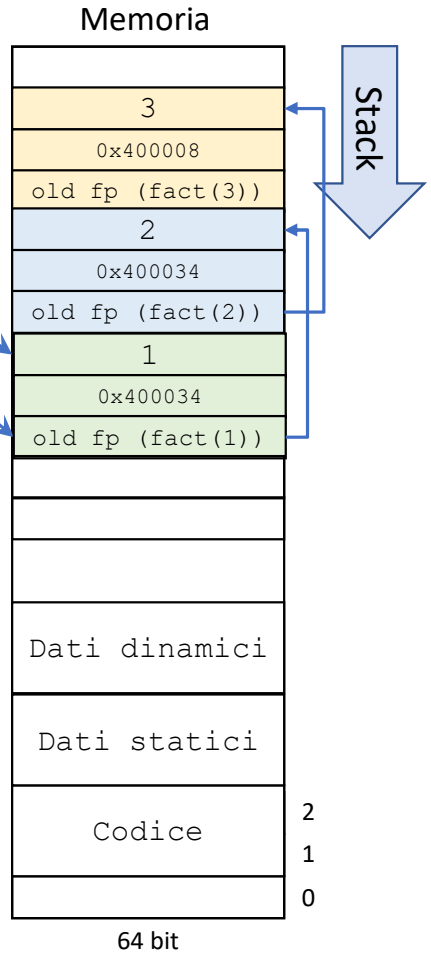


Esempio: calcolo del fattoriale

```
_start:
0x400000  li a0, 3 # salvo n in a0

0x400004  jal  ra, fact       # n in a0
0x400008  add  t0, a0, zero   # salva il risultato in t0

fact:
0x40000c  addi sp, sp, -24    # allocazione del call frame nello stack
0x400010  sd   a0, 16(sp)     # salvataggio di n nel call frame
0x400014  sd   ra, 8(sp)      # salvataggio dell'indirizzo di ritorno
0x400018  sd   fp, 0(sp)      # salvataggio del precedente frame pointer
0x40001c  addi  fp, sp, 16    # aggiornamento del frame pointer

0x400020  bne  a0, zero,  Ric # test fine ricorsione n!=0
0x400024  addi a0, zero, 1    # 0! = 1
0x400028  j    Fine

ric:                          # chiamata ricorsiva per il calcolo di (n-1)!
0x40002c  addi a0, a0, -1     # a0 <- (n - 1) parametro per fact(n-1)
0x400030  jal  fact           # chiama fact(n-1) -> risultato in a0
0x400034  ld   t0, 0(fp)      # t0 <- n
0x400038  mul  a0, a0, t0     # n! = (n-1)! x n

fine:
0x40003c  ld   fp, 0(sp)      # recupera il frame pointer
0x400040  ld   ra, 8(sp)      # recupera l'indirizzo di ritorno
0x400044  addi sp, sp, 24     # elimina il call frame dallo stack
0x400048  jr   ra             # ritorna al chiamante
```

a0	1
t0	
s0/fp	
sp	
ra	0x400034

Dopo la chiamata ricorsiva e la costruzione del record di attivazione di fact(1)

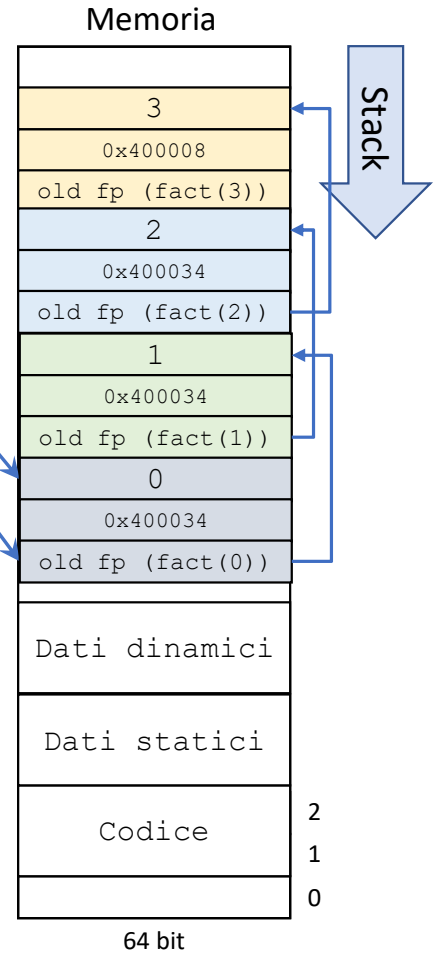


Esempio: calcolo del fattoriale

```
_start:
0x400000  li a0, 3 # salvo n in a0

0x400004  jal  ra, fact       # n in a0
0x400008  add  t0, a0, zero   # salva il risultato in t0

fact:
0x40000c  addi sp, sp, -24    # allocazione del call frame nello stack
0x400010  sd   a0, 16(sp)     # salvataggio di n nel call frame
0x400014  sd   ra, 8(sp)      # salvataggio dell'indirizzo di ritorno
0x400018  sd   fp, 0(sp)      # salvataggio del precedente frame pointer
0x40001c  addi  fp, sp, 16    # aggiornamento del frame pointer

0x400020  bne  a0, zero,  Ric # test fine ricorsione n!=0
0x400024  addi a0, zero, 1    # 0! = 1
0x400028  j    Fine

ric:                          # chiamata ricorsiva per il calcolo di (n-1)!
0x40002c  addi a0, a0, -1     # a0 <- (n - 1) parametro per fact(n-1)
0x400030  jal  fact           # chiama fact(n-1) -> risultato in a0
0x400034  ld   t0, 0(fp)      # t0 <- n
0x400038  mul  a0, a0, t0     # n! = (n-1)! x n

fine:
0x40003c  ld   fp, 0(sp)      # recupera il frame pointer
0x400040  ld   ra, 8(sp)      # recupera l'indirizzo di ritorno
0x400044  addi sp, sp, 24     # elimina il call frame dallo stack
0x400048  jr   ra             # ritorna al chiamante
```

a0	0
t0	
s0/fp	
sp	
ra	0x400034

Dopo la chiamata ricorsiva e la costruzione del record di attivazione di fact(0)

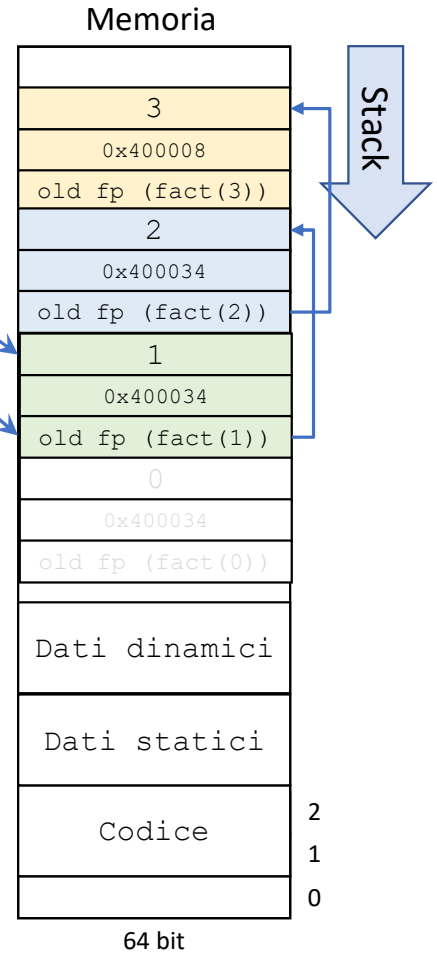


Esempio: calcolo del fattoriale

```
_start:
0x400000  li a0, 3 # salvo n in a0

0x400004  jal  ra, fact       # n in a0
0x400008  add  t0, a0, zero   # salva il risultato in t0

fact:
0x40000c  addi sp, sp, -24    # allocazione del call frame nello stack
0x400010  sd   a0, 16(sp)     # salvataggio di n nel call frame
0x400014  sd   ra, 8(sp)      # salvataggio dell'indirizzo di ritorno
0x400018  sd   fp, 0(sp)      # salvataggio del precedente frame pointer
0x40001c  addi  fp, sp, 16    # aggiornamento del frame pointer

0x400020  bne  a0, zero,  Ric # test fine ricorsione n!=0
0x400024  addi a0, zero, 1    # 0! = 1
0x400028  j    Fine

ric:                          # chiamata ricorsiva per il calcolo di (n-1)!
0x40002c  addi a0, a0, -1     # a0 <- (n - 1) parametro per fact(n-1)
0x400030  jal  fact           # chiama fact(n-1) -> risultato in a0
0x400034  ld   t0, 0(fp)      # t0 <- n
0x400038  mul  a0, a0, t0     # n! = (n-1)! x n

fine:
0x40003c  ld   fp, 0(sp)      # recupera il frame pointer
0x400040  ld   ra, 8(sp)      # recupera l'indirizzo di ritorno
0x400044  addi sp, sp, 24     # elimina il call frame dallo stack
0x400048  jr   ra             # ritorna al chiamante
```

a0	1
t0	
s0/fp	
sp	
ra	0x400034

n=0 e quindi la funzione ritorna 1, deallocando il record di attivazione

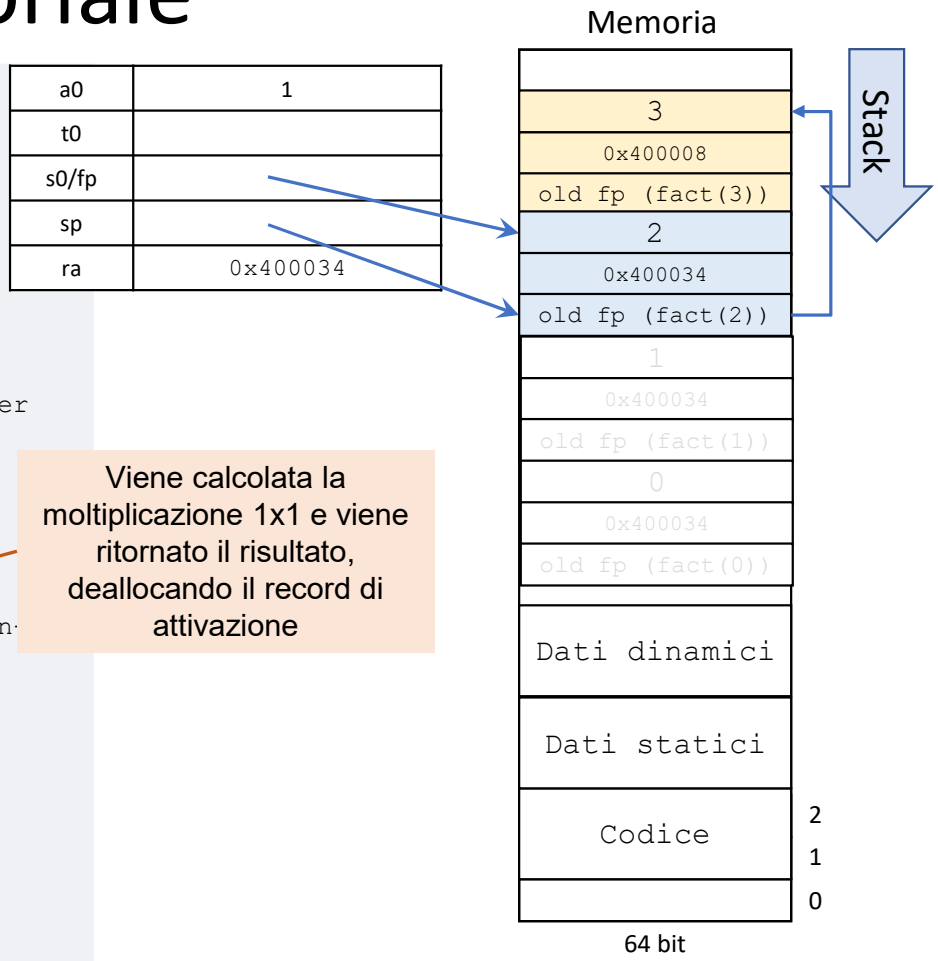


Esempio: calcolo del fattoriale

```
_start:
0x400000  li a0, 3 # salvo n in a0

0x400004  jal  ra, fact       # n in a0
0x400008  add  t0, a0, zero   # salva il risultato in t0

fact:
0x40000c  addi sp, sp, -24    # allocazione del call frame nello stack
0x400010  sd   a0, 16(sp)     # salvataggio di n nel call frame
0x400014  sd   ra, 8(sp)      # salvataggio dell'indirizzo di ritorno
0x400018  sd   fp, 0(sp)      # salvataggio del precedente frame pointer
0x40001c  addi  fp, sp, 16    # aggiornamento del frame pointer

0x400020  bne  a0, zero,  Ric # test fine ricorsione n!=0
0x400024  addi a0, zero, 1    # 0! = 1
0x400028  j    Fine

ric:                          # chiamata ricorsiva per il calcolo di (n-
0x40002c  addi a0, a0, -1     # a0 <- (n - 1) parametro per fact(n-1)
0x400030  jal  fact           # chiama fact(n-1) -> risultato in a0
0x400034  ld   t0, 0(fp)      # t0 <- n
0x400038  mul  a0, a0, t0     # n! = (n-1)! x n

fine:
0x40003c  ld   fp, 0(sp)      # recupera il frame pointer
0x400040  ld   ra, 8(sp)      # recupera l'indirizzo di ritorno
0x400044  addi sp, sp, 24     # elimina il call frame dallo stack
0x400048  jr   ra             # ritorna al chiamante
```

a0	1
t0	
s0/fp	
sp	
ra	0x400034

Viene calcolata la moltiplicazione 1x1 e viene ritornato il risultato, deallocando il record di attivazione



Esempio: calcolo del fattoriale

```
_start:
0x400000  li a0, 3 # salvo n in a0

0x400004  jal  ra, fact       # n in a0
0x400008  add  t0, a0, zero   # salva il risultato in t0

fact:
0x40000c  addi sp, sp, -24    # allocazione del call frame nello stack
0x400010  sd   a0, 16(sp)     # salvataggio di n nel call frame
0x400014  sd   ra, 8(sp)      # salvataggio dell'indirizzo di ritorno
0x400018  sd   fp, 0(sp)      # salvataggio del precedente frame pointer
0x40001c  addi  fp, sp, 16    # aggiornamento del frame pointer

0x400020  bne  a0, zero,  Ric # test fine ricorsione n!=0
0x400024  addi a0, zero, 1    # 0! = 1
0x400028  j    Fine

ric:                          # chiamata ricorsiva per il calcolo di (n-
0x40002c  addi a0, a0, -1     # a0 <- (n - 1) parametro per fact(n-1)
0x400030  jal  fact           # chiama fact(n-1) -> risultato in a0
0x400034  ld   t0, 0(fp)      # t0 <- n
0x400038  mul  a0, a0, t0     # n! = (n-1)! x n

fine:
0x40003c  ld   fp, 0(sp)      # recupera il frame pointer
0x400040  ld   ra, 8(sp)      # recupera l'indirizzo di ritorno
0x400044  addi sp, sp, 24     # elimina il call frame dallo stack
0x400048  jr   ra             # ritorna al chiamante
```

a0	2
t0	
s0/fp	
sp	
ra	0x400008

Viene calcolata la moltiplicazione 1x2 e viene ritornato il risultato, deallocando il record di attivazione

Esempio: calcolo del fattoriale

```
_start:
0x400000  li a0, 3 # salvo n in a0

0x400004  jal  ra, fact       # n in a0
0x400008  add  t0, a0, zero   # salva il risultato in t0

fact:
0x40000c  addi sp, sp, -24    # allocazione del call frame nello stack
0x400010  sd   a0, 16(sp)     # salvataggio di n nel call frame
0x400014  sd   ra, 8(sp)      # salvataggio dell'indirizzo di ritorno
0x400018  sd   fp, 0(sp)      # salvataggio del precedente frame pointer
0x40001c  addi  fp, sp, 16    # aggiornamento del frame pointer

0x400020  bne  a0, zero,  Ric # test fine ricorsione n!=0
0x400024  addi a0, zero, 1    # 0! = 1
0x400028  j    Fine

ric:                          # chiamata ricorsiva per il calcolo di (n-
0x40002c  addi a0, a0, -1     # a0 <- (n - 1) parametro per fact(n-1)
0x400030  jal  fact           # chiama fact(n-1) -> risultato in a0
0x400034  ld   t0, 0(fp)      # t0 <- n
0x400038  mul  a0, a0, t0     # n! = (n-1)! x n

fine:
0x40003c  ld   fp, 0(sp)      # recupera il frame pointer
0x400040  ld   ra, 8(sp)      # recupera l'indirizzo di ritorno
0x400044  addi sp, sp, 24     # elimina il call frame dallo stack
0x400048  jr   ra             # ritorna al chiamante
```

a0	6
t0	
s0/fp	
sp	
ra	

Viene calcolata la moltiplicazione 2x3 e viene ritornato il risultato al main, deallocando il record di attivazione

Riassunto dei formati delle istruzioni RISC-V

Nome (dimensione del campo)	Campi						Commenti
	7 bit	5 bit	5 bit	3 bit	5 bit	7 bit	
Tipo R	funz7	rs2	rs1	funz3	rd	codop	Istruzioni aritmetiche
Tipo I	Immediato[11:0]		rs1	funz3	rd	codop	Istruzioni di caricamento dalla memoria e aritmetica con costanti
Tipo S	immed[11:5]	rs2	rs1	funz3	immed[4:0]	codop	Istruzioni di trasferimento alla memoria (store)
Tipo SB	immed[12, 10:5]	rs2	rs1	funz3	immed[4:1,11]	codop	Istruzioni di salto condizionato
Tipo UJ	immediato[20, 10:1, 11, 19:12]				rd	codop	Istruzioni di salto incondizionato
Tipo U	immediato[31:12]				rd	codop	Formato caricamento stringhe di bit più significativi

Figura 2.19 Formati delle istruzioni RISC-V.

2
1
0