

معرفی کد میانی تسلنگ

در این مستند کد میانی تسلنگ (TSIR) معرفی می‌شود.

ساختار برنامه‌ها

هر فایل TSIR از یک یا بیشتر تابع تشکیل می‌شود. ماشین مجازی تسلنگ (TSVM) که کد میانی تسلنگ را اجرا می‌کند، اجرای یک فایل را از تابعی به نام main آغاز می‌نماید. در هر تابع مجموعه‌ای از رجیسترهای محلی در دسترس هستند. نام هر رجیستر از چسباندن حرف «r» با شماره‌ی رجیستر حاصل می‌شود. برای مثال r5 رجیستر پنجم یک تابع است. یک تابع با استفاده از کلمه‌ی کلیدی proc به شکل زیر تعریف می‌شود.

```
proc main
  mov  r0, 0
  ret
```

یک تابع می‌تواند تعدادی ورودی دریافت نماید. در هنگام فراخوانی یک تابع با عملگر call، مقدار رجیسترهایی که بعد از نام تابع مشخص می‌شوند به ترتیب در رجیسترهای r0، r1 و ... تابع فراخوانی شده کپی می‌شوند. در هنگام برگشت از تابع با عملگر ret، مقداری که در تابع در رجیستر r0 قرار داده شده است در ورودی اول (اولین رجیستر بعد از نام تابع در عملگر call) کپی می‌شود. برای نمونه در تابع sum3 مجموع سه عدد محاسبه می‌گردد.

```
proc sum3
  add  r0, r0, r1
  add  r0, r0, r2
  ret

proc main
  call iget, r3
  call iget, r1
  call iget, r2
  call sum3, r3, r1, r2
  call iput, r3
  mov  r0, 0
  ret
```

عملگرها

عملگرهای حسابی و مقایسه‌ای TSIR در جدول زیر نمایش داده می‌شوند.

دستور	توضیح
<code>mov r1, r2</code>	مقدار رجیستر <code>r2</code> را به <code>r1</code> می‌ریزد.
<code>mov r1, 5</code>	رجیستر <code>r1</code> را برابر عدد ۵ قرار می‌دهد.
<code>add r1, r2, r3</code>	حاصل جمع <code>r2</code> و <code>r3</code> را در <code>r1</code> قرار می‌دهد.
<code>sub r1, r2, r3</code>	حاصل تفریق <code>r3</code> از <code>r2</code> را در <code>r1</code> قرار می‌دهد.
<code>mul r1, r2, r3</code>	حاصل ضرب <code>r2</code> و <code>r3</code> را در <code>r1</code> قرار می‌دهد.
<code>div r1, r2, r3</code>	حاصل تقسیم <code>r2</code> بر <code>r3</code> را در <code>r1</code> قرار می‌دهد.
<code>mod r1, r2, r3</code>	باقیمانده‌ی تقسیم <code>r2</code> بر <code>r3</code> را در <code>r1</code> قرار می‌دهد.
<code>cmp= r1, r2, r3</code>	در صورتی که مقدار <code>r2</code> و <code>r3</code> برابر باشد، مقدار <code>r1</code> به یک و در غیر این صورت به صفر تغییر می‌کند.
<code>cmp< r1, r2, r3</code>	در صورتی که <code>r2</code> کوچک‌تر از <code>r3</code> باشد، مقدار <code>r1</code> به یک و در غیر این صورت به صفر تغییر می‌کند.
<code>cmp> r1, r2, r3</code>	در صورتی که <code>r2</code> بزرگ‌تر از <code>r3</code> باشد، مقدار <code>r1</code> به یک و در غیر این صورت به صفر تغییر می‌کند.
<code>cmp<= r1, r2, r3</code>	در صورتی که <code>r2</code> کوچک‌تر یا مساوی <code>r3</code> باشد، مقدار <code>r1</code> به یک و در غیر این صورت به صفر تغییر می‌کند.
<code>cmp>= r1, r2, r3</code>	در صورتی که <code>r2</code> بزرگ‌تر یا مساوی <code>r3</code> باشد، مقدار <code>r1</code> به یک و در غیر این صورت به صفر تغییر می‌کند.

عملگرهای پرش TSIR در جدول زیر نمایش داده می‌شوند.

دستور	توضیح
<code>jmp dst</code>	پرش به آدرس <code>dst</code> انجام می‌شود.
<code>jz r1, dst</code>	در صورتی که مقدار <code>r1</code> صفر باشد، پرش به آدرس <code>dst</code> انجام می‌شود.
<code>jnz r1, dst</code>	در صورتی که مقدار <code>r1</code> غیر صفر باشد، پرش به آدرس <code>dst</code> انجام می‌شود.

سایر عملگرهای TSIR در جدول زیر نمایش داده می‌شوند.

دستور	توضیح
<code>call func, r5, r9</code>	تابع <code>func</code> با ورودی‌های داده شده فراخوانی می‌شود.
<code>ret</code>	اجرای تابع خاتمه می‌یابد.
<code>ld r1, r2</code>	محتویات آدرسی که <code>r2</code> به آن اشاره می‌کند در <code>r1</code> ریخته می‌شود.
<code>st r1, r2</code>	مقدار <code>r1</code> در آدرسی که <code>r2</code> به آن اشاره می‌کند نوشته می‌شود.
<code>nop</code>	عملی انجام نمی‌دهد.

توابع داخلی

توابع داخلی TSVM در جدول زیر نمایش داده می‌شوند.

دستور	توضیح
<code>call iget, r1</code>	یک عدد از ورودی استاندارد می‌خواند و بر می‌گرداند.
<code>call iput, r1</code>	عدد داده شده را در خروجی استاندارد می‌نویسد.
<code>call mem, r1</code>	به مقدار خواسته شده حافظه تخصیص می‌دهد و آدرس حافظه را بر می‌گرداند.
<code>call rel, r1</code>	حافظه‌ی داده شده را آزاد می‌کند.

در نمونه‌ی زیر استفاده از توابع ورودی و خروجی نشان داده می‌شود.

```
proc main
  call iget, r0
  call iput, r0
  mov r0, 0
  ret
```

در نمونه‌ی زیر مدیریت و دسترسی به حافظه نشان داده می‌شود. در این نمونه، ۳۲ بایت حافظه تخصیص داده می‌شود و سپس آزاد می‌گردد.

```
proc main
  mov r1, 32
  call mem, r1
  mov r2, 5
  st r2, r1          # write r2 to *r1
  ld r3, r1          # read r3 from *r1
  call rel, r1
  mov r0, 0
  ret
```

نمونه‌های دیگر

در نمونه‌ی زیر استفاده از پرش شرطی نمایش داده می‌شود.

```
proc main
  call iget, r0
  call iget, r1
  cmp< r2, r0, r1
  jz r2, out
  mov r0, r1
out:
  call iput, r0
  mov r0, 0
  ret
```

محاسبه‌ی بزرگ‌ترین مقسوم علیه مشترک — آقای مشهدی

متوسط زمان اجرا: ۳۰/۵

طول برنامه: ۱۱

جواب اشتباه: ۰ از ۸

```
proc help
  call iget, r2
A:
  mod r3, r0, r2
  mov r0, r2
  mov r2, r3
  jnz r2, A
  ret
proc main
  call iget, r1
  call help, r1
  call help, r1
  call iput, r1
  ret
```

```
proc main
    call iget, r0
    call iget, r1
    call iget, r2
    call gcd2, r0, r1
    call gcd2, r0, r2
    call iput, r0
    ret
proc gcd2
    mod r2, r1, r0
    jz r2, OUT
    call gcd2, r2, r0
    mov r0, r2
OUT:
    ret
```

```
proc main
  call iget, r1
Q:
  call iget, r0
  cmp< r2, r1, r0
  jz r2, loop
  mov r3, r1
m:
  mov r1, r0
  mov r0, r3
  jz r3, s
loop:
  mod r3, r1, r0
  jmp m
s:
  jnz r4, out
  mov r4, 1
  jmp Q
out:
  call iput, r1
  ret
```

محاسبه‌ی بزرگ‌ترین مقسوم علیه مشترک — آقای یزدان‌پناه

متوسط زمان اجرا: ۵۲/۴

طول برنامه: ۱۶

جواب اشتباه: ۰ از ۸

```
proc main
  call iget, r0
  call iget, r1
  call iget, r2
  call gcd, r0, r1
  call gcd, r0, r2
  call iput, r0
  ret
proc gcd
begin:
  cmp> r2, r1, r0
  jz   r2, do
  mov  r2, r0
  mov  r0, r1
  mov  r1, r2
do:
  jz   r1, end
  mod  r0, r0, r1
  jmp  begin
end:
  ret
```



```
proc main
  call iget, r1
  call iget, r2
  call iget, r3
  call gcd, r1, r2
  call gcd, r1, r3
  call iput, r1
  ret
proc gcd
  cmp> r4, r0, r1
  jz r4, cnt
  mov r5, r0
  mov r0, r1
  mov r1, r5
cnt:
loop:
  mod r6, r1, r0
  jz r6, out
  mov r1, r0
  mov r0, r6
  jmp loop
out:
  ret
```

```
proc main
  call iget, r1
  call iget, r2
  call iget, r3
  mov  r0, 0
  mov  r4, 0
first:
  cmp== r5, r1, r0
  jz   r5, bnext
  mov  r1, r2
  jnz  r5, next
bnext:
  cmp== r5, r2, r0
  jnz  r5, next
  mov  r5, r2
  mod  r2, r1, r2
  mov  r1, r5
  jmp  first
next:
  jnz  r4, finish
  mov  r4, 1
  mov  r2, r3
  jmp  first
finish:
  call iput, r1
  ret
```

```

proc main
  mov  r0, 8
  mov  r1, 8
  mov  r2, 8
  call mem, r0
  call mem, r1
  call mem, r2
  call iget, r3
  call iget, r4
  call iget, r5
  st   r3, r0
  st   r4, r1
  st   r5, r2
  call findmin, r0, r1
  call gcd, r0, r1
  call findmin, r0, r2
  call gcd, r0, r2
  ld   r6, r0
  call iput, r6
  ret

proc findmin
  ld   r2, r0
  ld   r3, r1
  cmp<= r4, r2, r3
  jnz  r4, out1
  st   r3, r0
  st   r2, r1
out1:
  ret

proc gcd
  ld   r2, r0
  ld   r3, r1
op:
  mod  r4, r3, r2
  jz   r4, out2
  mov  r3, r2
  mov  r2, r4
out2:
  st   r2, r0
  ret

```