MÃ TRUNG GIAN CHO NGÔN NGỮ SIMPLE C

Giới thiệu

Tài liệu này "mô tả" "một" *mã trung gian* cho ngôn ngữ *Simple C*, tạm gọi là *HIR*. Mã trung gian này có hướng đến kiến trúc máy *MIPS* nhưng vẫn cố gắng giữ được tính độc lập để dễ dàng biên dịch sang mã hợp ngữ (hay mã máy) trên các kiến trúc máy khác nhau. Chìa khóa là *tuyến tính hóa* (phẳng hóa) mã Simple C nhưng vẫn giữ được các đặc trưng của Simple C. Mục đích của mã trung gian này là giúp giảm thiểu các khó khăn khi biên dịch mã Simple C sang mã cấp thấp. Đặc điểm chính của nó là:

- Dựa trên mã trung gian 3 địa chỉ.
- Chỉ số hóa và cụ thể hóa các loại biến.
- Phẳng hóa các biểu thức và câu lệnh.
- Che dấu việc sử dụng bộ nhớ, hỗ trợ cấp phát và truy xuất mảng.
- Tự động quản lý ngoại lệ.
- Hỗ trợ việc gọi, thực thi và trả về từ các hàm.

Ngoài ra, hiện tại, ngôn ngữ trung gian này chỉ có một kiểu dữ liệu là số nguyên có dấu 4-byte (cùng với các hằng chuỗi trong lệnh xuất). Ngôn ngữ cũng ngầm định hỗ trợ kiểu luận lý với giá trị TRUE là 1 và FALSE là 0.

Vì mã trung gian này không là mã máy nên không chạy trực tiếp trên các kiến trúc máy. Nó phải được một *động cơ chạy mã trung gian* thi hành. Đây là chương trình thông dịch mã trung gian. Hoặc là mã trung gian sẽ được biên dịch tiếp ra mã hợp ngữ, rồi ra mã máy của một kiến trúc máy (nhất là MIPS) rồi thi hành trực tiếp trên máy này.

Cú pháp

Toàn bộ mã trung gian được chứa duy nhất trong một file tương ứng với file mã Simple C. File chứa mã trung gian là file văn bản được tổ chức thành nhiều dòng. Cho phép các dòng trống và ghi chú trên một dòng từ dấu #. Có thể dùng các khoảng trắng (space và tab) để định dạng mã. Các thành phần từ vựng của mã trung gian gồm:

- *Tên lệnh*: là dãy các kí tự, dùng để xác định lệnh hay khai báo. Các khai báo và tập lệnh của ngôn ngữ được mô tả bên dưới. Ví dụ: `str`, `add`, …

- *Tên hàm*: có dạng *name*, với *name* là tên của hàm tương ứng ở mã Simple C. Ví dụ: `main`, `fact`, …

- *Nhãn*: có dạng ~*x*, với *x* là số thứ tự. Nhãn được dùng để "đánh dấu" lệnh, dùng trong các lệnh nhảy. Nhãn cũng có phạm vi toàn cục nên *x* giúp định danh duy nhất nhãn. Ví dụ: `~0`, `~1`, …

- *Biến cục bộ*: có dạng @*x*[*_name*], với *x* là số thứ tự, *name* (nếu có) là tên của biến tương ứng ở mã Simple C. Biến cục bộ là biến được khai báo bên trong thân hàm trong mã Simple C. Ví dụ: `@0`, `@1_N`, …

- *Biến toàn cục*: có dạng $*x*[*_name*] với *x* là số thứ tự, *name* (nếu có) là tên của biến toàn cục tương ứng ở mã Simple C. Biến toàn cục là biến được khai báo bên ngoài các hàm trong mã Simple C. Ví dụ `$0`, `$1_a`, …

- *Tham số*: có dạng %*x*[*_name*], với *x* là số thứ tự, *name* (nếu có) là tên của tham số tương ứng ở mã Simple C. Ví dụ `%0`, `%1_N`, …

- *Biến tạm*: có dạng &*x*, với *x* là số thứ tự. Biến tạm là biến dùng tạm bên trong hàm để chứa các kết quả tính toán trung gian của biểu thức, câu lệnh. Biến tạm không có trong mã Simple C nhưng có trong mã trung gian do việc tuyến tính hóa các biểu thức, câu lệnh. Ví dụ: `&0`, `&1`, …

- *Hằng chuỗi*: "…" dùng liên kết với tham chiếu hằng chuỗi để phục vụ cho lệnh xuất. Đây là các hằng chuỗi tương ứng trong mã Simple C. Hằng chuỗi này cho phép dùng các kí tự thoát như trong Simple C, chẳng hạn "\n" là xuống dòng.

- *Tham chiếu hằng chuỗi*: có dạng ?*x*, với *x* là số thứ tự khai báo của hằng chuỗi.

- *Hằng*: là hằng số nguyên có dấu (độ lớn 4 byte). Ví dụ: 0, -1, 10, …

- *Dấu câu*: mã trung gian chỉ dùng hai dấu câu là dấu ',' dùng để phân cách các đối số của lệnh hoặc khai báo và dấu ':' dùng trong khai báo nhãn.

Các số thứ tự được tính từ 0. Biến cục bộ, biến toàn cục, tham số và biến tạm được gọi chung là *biến*. *Giá trị* của biến là giá trị chứa trong biến tại thời điểm biến được truy xuất. Giá trị của hằng là bản thân hằng đó. *Gán* giá trị cho biến là làm cho biến chứa giá trị được gán.

Các khai báo và lệnh của ngôn ngữ trung gian đều được viết trên một dòng với cú pháp chung là (trừ khai báo nhãn):

```
name opr1 [, opr2] [, opr3]
```

Trong đó: `name` là tên của khai báo hay lệnh; `opr1`, `opr2`, `opr3` là các đối số của khai báo hay lệnh; tùy khai báo hay lệnh mà có ít nhất là 1 và nhiều nhất là 3 đối số. Có ít nhất một kí tự trắng giữa `name` và `opr1`. Các `opr2`, `opr3` nếu có phải phân cách với đối số trước đó bằng kí tự ',' (và có thể có các kí tự trắng).

Bố cục và khai báo

Toàn bộ mã trung gian trong file mã trung gian được gọi là *chương trình*. Chương trình bắt đầu bằng các khai báo hằng chuỗi nếu có. Mỗi khai báo hằng chuỗi có cú pháp:

```
str strconst
```

với `strconst` là hằng chuỗi. Mọi hằng chuỗi phải được khai báo liên tục ngay đầu chương trình. Hằng chuỗi được dùng kèm với tham chiếu hằng chuỗi ?*x*. *Hằng chuỗi tương ứng* với tham chiếu này là hằng chuỗi được khai báo ở lần thứ *x*. Tham chiếu hằng chuỗi chỉ có thể được dùng trong lệnh xuất.

Kế tiếp là khai báo tên *hàm vào* của chương trình và số lượng biến toàn cục. Hàm vào là hàm mà chương trình bắt đầu chạy từ đó. Trong Simple C thì qui ước hàm vào là `void/int main()`. Cú pháp khai báo:

```
entry name, n
```

với `name` là tên của hàm vào; `n` là hằng cho biết số lượng biến toàn cục của chương trình. Nếu chương trình không có biến toàn cục thì *n* là 0.

Kế tiếp là khai báo và lệnh của các hàm trong chương trình. Lưu ý: có ít nhất một hàm, đó chính là hàm vào. Cú pháp khai báo hàm như sau:

```
func name

funci numvar, numtemp

# các lệnh của hàm

efunc name
```

Trong đó: `name` là tên của hàm đang khai báo; `numvar`, `numtemp` là số lượng biến cục bộ và số lượng biến tạm dùng trong các lệnh của hàm.

Lưu ý: **`func`** và **`funci`** được xem là khai báo nhưng **`efunc`** được xem là lệnh vì về mặt ngữ nghĩa nó tương đương với lệnh trở về không có giá trị từ hàm.

Xen giữa các lệnh có thể có các *khai báo nhãn*. Mỗi khai báo nhãn được viết trên một dòng với cú pháp:

```
label:
```

với `label` là tên nhãn; có thể có khoảng trắng giữa nhãn và dấu hai chấm. Khai báo nhãn được dùng để "đánh dấu" lệnh và dùng kèm với các lệnh nhảy. Lệnh kế ngay sau khai báo nhãn được gọi là *lệnh sau nhãn*.

Các lệnh

Số học, luận lý và quan hệ

Các lệnh thực hiện các phép toán số học, luận lý, quan hệ hai ngôi có cú pháp:

```
name result, opr1, opr2
```

Trong đó: `name` là tên lệnh; `result` là biến; `opr1`, `opr2` là biến hoặc hằng. Các lệnh này thực hiện phép toán tương ứng trên giá trị của các toán hạng `opr1` và `opr2`, sau đó gán kết quả cho `result`.

Các phép toán 2 ngôi và tên lệnh tương ứng là:

- *Phép toán số học*: cộng (**`add`**), trừ (**`sub`**), nhân (**`mult`**), chia lấy nguyên (**`div`**), chia lấy dư (**`mod`**). Động cơ chạy sẽ tự động quản lý ngoại lệ "*chia cho 0*".
- *Phép toán logic*: và (**`and`**), hoặc (**`or`**). Các toán hạng chỉ có thể có giá trị là TRUE (1) hoặc FALSE (0). Kết quả của phép toán là TRUE hoặc FALSE.
- *Phép toán quan hệ*: lớn hơn (**`gt`**), lớn hơn hoặc bằng (**`gte`**), nhỏ hơn (**`lt`**), nhỏ hơn hoặc bằng (**`lte`**), bằng (**`eq`**), không bằng (**`neq`**). Kết quả của phép toán là TRUE hoặc FALSE.

Phép toán số học một ngôi duy nhất là phép lấy đối, cú pháp:

```
comp result, opr
```

Trong đó: `result` là biến; `opr` là biến hoặc hằng. Đối của giá trị của `opr` sẽ được gán vào `result`.

Phép toán luận lý một ngôi duy nhất là phép lấy phủ định, cú pháp:

```
not result, opr
```

Trong đó: `result` là biến; `opr` là biến hoặc hằng với giá trị là TRUE hoặc FALSE. Phủ định của giá trị của `opr` sẽ được gán vào `result`.

Gán

Cú pháp:

```
move result, value
```

Trong đó: `result` là biến; `value` là biến hoặc hằng. Giá trị của `value` sẽ được gán vào `result`.

Nhập và xuất

Mã trung gian hỗ trợ việc nhập và xuất các số nguyên.

Lệnh nhập:

```
read result
```

với `result` là biến. Động cơ chạy sẽ chờ người dùng nhập một số nguyên, sau đó gán số nguyên này vào `result`.

Lệnh xuất:

```
write value
```

với `value` là biến hoặc hằng hoặc tham chiếu hằng chuỗi. Nếu là tham chiếu hằng chuỗi thì hằng chuỗi tương ứng sẽ được xuất ra, nếu không thì giá trị tương ứng sẽ được xuất ra.

Nhảy

Lệnh nhảy vô điều kiện:

```
jump label
```

với `label` là nhãn. Nhảy tới thực thi lệnh sau `label`.

Nhảy có điều kiện:

```
jt value, label
```

hoặc

```
jf value, label
```

với `value` là biến hoặc hằng, `label` là nhãn. Nhảy tới thực thi lệnh sau `label` nếu `value` có giá trị TRUE (với **`jt`**) hoặc có giá trị FALSE (với **`jf`**); nếu không thì tiếp tục thực thi lệnh kế tiếp.

Các lệnh nhảy dựa trên so sánh:

```
name value1, value2, label
```

Trong đó: `name` là tên lệnh; `value1`, `value2` là biến hoặc hằng; `label` là nhãn. Nhảy đến lệnh sau `label` nếu `value1` và `value2` thỏa mãn phép so sánh tương ứng nếu không thì tiếp tục thực thi lệnh kế tiếp.

Các phép so sánh và tên lệnh tương ứng là: bằng (**`jeq`**), không bằng (**`jneq`**), nhỏ hơn (**`jlt`**), nhỏ hơn hoặc bằng (**`jlte`**).

Lưu ý: mặc dù các nhãn có phạm vi toàn cục nhưng không cho phép các lệnh nhảy nhảy đến một nhãn ở một hàm khác với hàm chứa lệnh nhảy.

Mảng

Cấp phát mảng:

```
arra array, length
```

Cấp pháp mảng có `length` phần tử và cho `array` trỏ đến mảng đó. Khi đó giá trị của `array` được gọi là *con trỏ trỏ đến mảng* đã được cấp phát.

Lấy giá trị của một phần tử trong mảng:

```
arrg result, array, index
```

Lấy giá trị của phần tử có chỉ số `index` của mảng do `array` trỏ đến và gán vào `result`.

Đặt giá trị của một phần tử trong mảng:

```
arrs array, index, value
```

Lấy giá trị của `value` gán vào phần tử có chỉ số `index` của mảng do `array` trỏ đến.

Trong các lệnh trên thì: `array` là biến; `index` là biến hoặc hằng; `result` là biến; `value` là biến hoặc hằng; `length` là biến hoặc hằng.

Chỉ số mảng tính từ 0 và động cơ chạy sẽ tự động quản lý ngoại lệ truy cập (lấy hoặc đặt) "*quá chỉ số mảng*".

Con trỏ

Mã trung gian HIR hiện chưa hỗ trợ con trỏ.

Hàm

Việc khai báo hàm đã được đề cập ở trên.

Việc gọi hàm bao gồm việc "đặt" các đối số của hàm và phát lệnh gọi hàm. Lệnh đặt đối số của hàm có cú pháp:

```
arg value, index
```

Trong đó: `value` là biến hoặc hằng; `index` là hằng. Giá trị của `value` sẽ được lấy làm đối số thứ `index` (tính từ 0). Các lệnh đặt đối số phải liên tục theo đúng số thứ tự từ thấp đến cao và liền ngay trước lệnh gọi hàm. Nếu hàm không có đối số thì không có lệnh đặt đối số nào cả trước lệnh gọi hàm.

Lệnh gọi hàm không có trả về giá trị:

```
call name, n
```

Trong đó: `name` là tên của hàm cần gọi; `n` là hằng cho biết số lượng đối số. Nhảy tới thực thi hàm tương ứng với các đối số đã được đặt trong các lệnh đặt đối số trước đó; đồng thời nhớ lệnh ngay sau lệnh gọi hàm để tiếp tục thực hiện khi trở về từ hàm được gọi. Nếu hàm không có đối số thì *n* là 0; nếu không, *n* phải phù hợp với các lệnh đặt đối số trước đó.

Lệnh gọi hàm có trả về giá trị:

```
callf result, name, n
```

Trong đó: *result* là biến; `name` là tên của hàm cần gọi; `n` là hằng cho biết số lượng đối số. Tương tự lệnh gọi hàm không có giá trị trả về nhưng ngay sau khi trở về từ hàm được gọi thì gán giá trị trả về cho `result`. Động cơ chạy tự động quản lý ngoại lệ "*không có giá trị trả về*".

Lệnh thoát khỏi hàm mà không trả về giá trị cho hàm:

```
ret name
```

với `name` là tên hàm. Lưu ý: `name` phải là tên hàm chứa lệnh trả về này.

Lệnh thoát khỏi hàm có trả về giá trị cho hàm:

```
retf name, value
```

Trong đó: `name` là tên hàm; `value` là biến hoặc hằng. Thoát khỏi hàm và trả về giá trị cho hàm là giá trị của `value`. Lưu ý: `name` phải là tên hàm chứa lệnh trả về này.

Thực thi chương trình

Động cơ chạy thực thi chương trình theo các bước sau:

1. Duyệt qua các khai báo hằng chuỗi; đồng thời nhớ thứ tự khai báo.
2. Đọc khai báo hàm vào.
3. Thực thi hàm vào.
4. Kết thúc chương trình khi trở về từ hàm vào.

Động cơ chạy thực thi một hàm theo các bước sau:

1. Nạp lệnh đầu tiên của hàm làm *lệnh hiện hành*.
2. Xét lệnh hiện hành:
    - Nếu là **`efunc`** thì trở về không giá trị.
    - Nếu là **`ret`** thì trở về không giá trị, là **`retf`** thì trở về với giá trị trả về tương ứng.
    - Nếu là **`call`**/**`callf`** thì thực thi hàm tương ứng; khi trở về, nạp lệnh kế tiếp làm lệnh hiện hành.
    - Nếu là lệnh nhảy thì tùy điều kiện mà nạp lệnh kế tiếp hay nạp lệnh sau lệnh nhảy làm lệnh hiện hành.
    - Các lệnh khác thì thực thi và nạp lệnh kế tiếp làm lệnh hiện hành.
3. Quay lại Bước 2.

Phần này cùng với các phần trước "góp phần" mô tả ngữ nghĩa của mã trung gian này. Nếu mã trung gian được biên dịch sang mã cấp thấp của các kiến trúc máy thì mã cấp thấp cùng với kiến trúc máy phải đảm bảo ngữ nghĩa này.

Ví dụ

Chương trình "Hello World!" kinh điển trong mã HIR

```
str "Hello World!\n"

entry main, 0

func main
```

```
funci 0, 0
     write ?0
efunc main
```

Một chương trình Simple C

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

     int f = fact(n - 1);

     return f * n;
}

void main()
{
     int n;

     cout << "Input n: ";
     cin >> n;
     cout << n << "! = " << fact(n);
}
```

Và mã trung gian HIR tương ứng

```
str "Input n: "
str "! = "

entry main, 0

func fact
funci 1, 1
     jneq %0_n, 0, ~0
     retf fact, 1
     ~0:
     sub &0, %0_n, 1
     arg &0, 0
     callf @0_f, fact, 1
     mult &0, @0_f, %0_n
```

```
     retf fact, &0
efunc fact

func main
funci 1, 1
     write ?0
     read @0_n
     write @0_n
     write ?1
     arg @0_n, 0
     callf &0, fact, 1
     write &0
efunc main
```

-- HẾT --