

Kỹ thuật dịch ngược

Ngày 25 tháng 11 năm 2016

**Lời cam đoan**

**Lời cảm ơn**

**Tóm tắt luận văn**

## Mục lục

|          |  |          |
|----------|--|----------|
| <b>1</b> | <b>Giới thiệu</b>  | <b>1</b> |
| 1.1      | Kỹ thuật dịch ngược và ứng dụng . . . . .  | 1        |
| 1.2      | Bài toán đặt ra . . . . .  | 1        |
| 1.3      | Cấu trúc luận văn . . . . .  | 2        |
| <b>2</b> | <b>Kiểm tra kiểu - Type checking</b>   | <b>3</b> |
| 2.1      | Mẫu khai báo biến byte và biến bit . . . . .                                     | 3        |
| 2.2      | Phân tích Reaching definitions . . . . .   | 3        |
| 2.3      | Kiểm tra cách sử dụng bộ biến trong toàn bộ chương trình và<br>sinh mã . . . . . | 4        |
| <b>3</b> | <b>Suy luận kiểu - Type inference</b>  | <b>6</b> |
| 3.1      | Phân tích Constant propagation . . . . .   | 6        |
| 3.2      | Quét các câu lệnh sử dụng biến bit . . . . .                                     | 7        |

## Danh sách hình vẽ

# 1 Giới thiệu

Chương này nhằm mục đích giới thiệu về bài toán sẽ được giải quyết trong luận văn và các khái niệm liên quan. Đầu chương sẽ nói về kỹ thuật dịch ngược, các ứng dụng của nó và những khó khăn trong quá trình dịch ngược. Phần tiếp theo trình bày bài toán đặt ra và các thách thức khi giải quyết bài toán. Phần cuối cùng tóm tắt cấu trúc của luận văn.

## 1.1 Kỹ thuật dịch ngược và ứng dụng

## 1.2 Bài toán đặt ra

Như đã đề cập ở phần trên, mục tiêu của luận văn là nghiên cứu về trình dịch ngược từ mã assembly lên mã cấp cao, các bài toán cần phải giải quyết và hiện thực giải pháp. Vì mã assembly cho kiến trúc máy khác nhau có những đặc điểm khác nhau, và đi cùng với đó là những vấn đề khác nhau cần giải quyết, nên giới hạn của luận văn sẽ là trình dịch ngược từ mã assembly 8051. Việc chọn kiến trúc máy 8051 là do 2 nguyên nhân sau:

- Chip 8051 đã xuất hiện trên thị trường từ lâu, hiện tại đã không còn được sản xuất. Tuy nhiên, vẫn còn nhiều hệ thống được chạy trên đây và cần phải chuyển đổi chúng sang một kiến trúc máy khác hiện đại hơn.
- Chip 8051 có một số đặc điểm khác biệt so với các con chip khác trên thị trường. Vì vậy việc dịch ngược từ mã 8051 sẽ gặp nhiều khó khăn hơn, vấn đề phải giải quyết phức tạp hơn.

Các đặc điểm khác biệt của 8051 gồm có:

- Trong khi hầu hết các kiến trúc máy khác sử dụng kiểu dữ liệu byte là kiểu dữ liệu nhỏ nhất, thì 8051 cho phép lập trình viên truy xuất tới mức bit trong một số thanh ghi và kèm theo đó là các câu lệnh xử lý bit. Tuy nhiên, các thanh ghi này của 8051 cũng có thể được truy xuất ở mức byte bình thường (xem ví dụ ở đoạn mã ??).
- Một số assembler của 8051 cho phép sử dụng tên biến. Biến này dùng để lưu các giá trị hằng số, hằng số này thường là địa chỉ một vùng nhớ kích thước 1 byte (trong luận văn này sẽ gọi tắt là biến byte) hoặc đại diện cho bit của thanh ghi (gọi tắt là biến bit). Khi lập trình, người ta thường sử dụng biến byte và biến bit này theo bộ, nghĩa là chỉ khi thanh ghi được load vào giá trị vùng nhớ quy định bởi biến byte, thì các biến bit cùng bộ mới được sử dụng (xem ví dụ ở đoạn mã ??).

Từ các đặc điểm trên, ta có thể thấy bài toán lớn nhất đặt ra trong luận văn này sẽ là tìm ra được mối liên hệ giữa biến byte và biến bit trong chương trình, lấy được các bộ biến byte và biến bit đúng. Có 2 cách để biết được điều này:

- Đưa ra quy định về việc khai báo biến byte và biến bit. Hiện nay, ở phần khai báo, các lập trình viên có thể khai báo các biến theo thứ tự tùy ý, và cũng không có quy định nào bắt buộc họ phải có phần comment chỉ rõ các biến byte và biến bit nào là cùng một bộ. Ta có thể đưa ra các mẫu khai báo cho biến byte và biến bit để trình dịch ngược có thể biết được các bộ biến bằng cách đọc theo mẫu mà không cần phân tích gì thêm. Tuy nhiên, sau khi đã xác định được các bộ biến này, cần có thêm một bước kiểm tra mã chương trình để đảm bảo rằng nguyên tắc sử dụng biến byte và biến bit được tuân thủ (Kiểm tra kiểu - Type checking). Giải pháp này có ưu điểm là đơn giản, dễ hiện thực nhưng gây bất tiện cho người dùng vì phải chuyển đổi bộ mã hiện tại về theo mẫu quy định.
- Dựa vào phân tích luồng dữ liệu của chương trình, tìm ra được địa chỉ vùng nhớ được load vào thanh ghi tại thời điểm sử dụng biến bit và từ đó suy ra biến byte cùng bộ với biến bit đó (Suy luận kiểu - Type inference). Với cách làm này, không cần phải thay đổi đoạn mã gốc. Tuy nhiên cách hiện thực sẽ phức tạp hơn nhiều vì có rất nhiều cách load dữ liệu vùng nhớ vào thanh ghi như: dùng trực tiếp hằng số, dùng trực tiếp biến byte, trung gian qua một thanh ghi khác... (xem ví dụ ở đoạn mã ??)

Cả hai giải pháp này đều được hiện thực trong từng giai đoạn của luận văn và sẽ được trình bày trong các chương tiếp theo.

### 1.3 Cấu trúc luận văn

Luận văn sẽ gồm 6 chương như sau:

- Chương 1: Giới thiệu về kỹ thuật dịch ngược, bài toán đặt ra trong luận văn và cấu trúc của luận văn
- Chương 2: Nêu lên một số kiến thức cơ bản và các nghiên cứu liên quan đến luận văn
- Chương 3: Trình bày giải pháp Kiểm tra kiểu - Type checking
- Chương 4: Trình bày giải pháp Suy luận kiểu - Type inference
- Chương 5: Đánh giá kết quả của luận văn thông qua các mẫu thử (testcase)
- Chương 6: Kết luận

## 2 Các kiến thức nền tảng và nghiên cứu liên quan

### 2.1 Cấu trúc của Boomerang

Ở giai đoạn thực tập tốt nghiệp, sau quá trình nghiên cứu một số trình dịch ngược hiện nay, người viết đã quyết định sử dụng trình dịch ngược Boomerang vì sử dụng nhiều kỹ thuật phân tích giúp chất lượng mã đầu ra cao và có cấu tạo kiểu module dễ thay đổi, thêm mới. Phần này sẽ giới thiệu về cấu trúc code của Boomerang, giúp ích cho việc trình bày các giải pháp của bài toán ở chương kế.

Về mặt tổng thể, Boomerang gồm có các phần sau:

Khi đọc vào một chương trình assembly, Boomerang sẽ lưu trữ chúng dưới cấu trúc sau:

Khi giải mã lên ngôn ngữ trung gian, cấu trúc Boomerang dùng để thể hiện là:

Prog là tương ứng với toàn bộ chương trình. Một Proc là một hàm, BasicBlock đại diện cho một khối cơ bản mà ở đó không có một câu lệnh rẽ nhánh nào (ví dụ như if, hoặc vòng lặp...). Statement là một câu lệnh và Expr là các biểu thức trong chương trình. Ngoài ra còn có các class đại diện cho kiểu dữ liệu.

Việc thực hiện các phân tích chủ yếu diễn ra tại Proc, vì vậy, các thay đổi trong luận văn này cũng chủ yếu được thực hiện bằng các hàm của Proc.

### 3 Kiểm tra kiểu - Type checking

Như đã giới thiệu ở phần trước, giải pháp Kiểm tra kiểu gồm có các bước sau:

- Đưa ra mẫu khai báo biến byte và biến bit. Chỉnh sửa chỉnh dịch ngược để đọc vào mẫu khai báo này và biết được những biến byte và biến bit nào cùng một bộ.
- Thực hiện phân tích dữ liệu để biết được giá trị của thanh ghi ACC tại một thời điểm nào đó có phải là giá trị của một vùng nhớ cố định hay không. Tại giai đoạn này của luận văn, kỹ thuật phân tích dữ liệu được sử dụng là *reaching definition*.
- Tại các câu lệnh sử dụng bit, nếu giá trị của thanh ghi ACC là của một vùng nhớ cố định, thì kiểm tra xem giá trị địa chỉ của vùng nhớ đó đang được lưu bởi biến nào. Nếu biến đó được khai báo cùng bộ với biến bit thì xem như tuân thủ đúng nguyên tắc sử dụng, nếu không thì báo lỗi.
- Nếu toàn bộ chương trình đều tuân thủ đúng nguyên tắc sử dụng biến byte - biến bit thì đưa các bộ biến byte - biến bit về dạng union ở ngôn ngữ cấp cao.

Chương này sẽ lần lượt trình bày các bước trên.

#### 3.1 Mẫu khai báo biến byte và biến bit

Mẫu khai báo biến byte và biến bit được quy định như sau: Đây là mẫu khai báo khi muốn gom các biến vào cùng một bộ, ngoài ra, chương trình vẫn chấp nhận việc khai báo riêng lẻ từng biến. Như vậy, cần chỉnh lại phần parser của trình dịch ngược để chấp nhận cấu trúc mới này. Trong trình dịch ngược, cần có một cấu trúc mới để lưu trữ thông tin của các bộ biến byte và biến bit này. Như vậy, sau giai đoạn parser, ta đã thu được bộ biến theo khai báo của người dùng. Tuy nhiên, nguyên tắc sử dụng bộ biến này là không bắt buộc trong quá trình lập trình 8051, và các assembler cũng không hề kiểm tra việc người dùng có tuân thủ nguyên tắc này không. Vì vậy, trước khi chuyển hoá các bộ biến này sang cấu trúc union ở mã đầu ra, ta cần phải kiểm tra đoạn mã của người dùng sử dụng các bộ biến như thế nào.

#### 3.2 Phân tích Reaching definitions

Mục đích của phân tích Reaching definitions là biết được ở một thời điểm của chương trình, các câu lệnh khai báo nào đang còn hiệu lực, hay nói cách khác là giá trị của các biến đang là gì. Như vậy, khi áp dụng vào trình dịch ngược, ta sẽ biết được tại thời điểm sử dụng biến bit, giá trị của thanh ghi ACC đang được định nghĩa như thế nào. Nếu đó là một câu lệnh load một vùng nhớ có địa chỉ quy định bởi biến byte, thì ta kiểm tra tiếp biến byte đó có cùng bộ với biến bit đang sử dụng hay không. Lưu ý là ta chỉ xét trường hợp thanh ghi ACC được load một giá trị vùng nhớ (biểu diễn ở ngôn ngữ cấp cao là pointer). Các bước



phân tích reaching definitions áp dụng vào cấu trúc dữ liệu hiện tại của trình dịch ngược như sau:

Tuy nhiên, phân tích Reaching definitions chỉ cho biết được câu lệnh khai báo có hiệu lực của một biến tại một thời điểm chương trình, chứ không cho biết được giá trị thực sự của biến đó. Nếu về phải của câu lệnh khai báo này chỉ đơn giản là pointer của một biến byte, thì ta sẽ dễ dàng kiểm tra được. Nhưng ngoài ra, biểu thức nằm trong dấu pointer khi dịch ngược từ mã assembly 8051 lên có thể là các trường hợp sau đây:

- Một biểu thức có hai vế, các vế của biểu thức có thể là một biến, một thanh ghi hoặc một hằng số.
- Một thanh ghi, giá trị của thanh ghi có thể được khai báo ở các câu lệnh trước đó.

Với phương pháp Reaching definitions, ta sẽ không thể xử lý được với các trường hợp phức tạp hơn như trên. Một phân tích khác sẽ được áp dụng trong giai đoạn tiến hành giải pháp Suy luận kiểu. Còn với giai đoạn sơ khai này của luận văn, ta sẽ giả thuyết rằng ở đoạn mã đầu vào chỉ có các câu lệnh gán vùng nhớ đơn giản cho ACC với một biến byte duy nhất.

### 3.3 Kiểm tra cách sử dụng bộ biến trong toàn bộ chương trình và sinh mã

Sau khi đã có thông tin về reaching definitions ở tất cả các điểm của chương trình, ta sẽ chuyển sang kiểm tra các ở các câu lệnh sử dụng bit.

Quá trình kiểm tra gồm các bước sau:

Nếu ở bất cứ câu lệnh sử dụng biến bit nào, nguyên tắc đưa ra bị vi phạm, thì trình dịch ngược sẽ hiện thông báo lỗi trên console cho người dùng. Sau đó, đoạn mã kiểm tra lỗi vẫn tiếp tục chạy để kiểm tra các câu lệnh khác nhằm tạo thuận lợi cho người dùng trong quá trình sửa lỗi chương trình đầu vào. Tuy nhiên, nếu có lỗi thì trình dịch ngược sẽ không sinh mã đầu ra vì sẽ không thể xử lý được sinh các union như thế nào.

Sau quá trình kiểm tra lỗi, trình dịch ngược sẽ tiến hành sinh ra các union ở mã đầu ra và thay thế các biến bit trong chương trình bằng các biểu thức truy xuất union, cũng như bỏ các câu lệnh gán cho ACC và thay thế biến ACC bằng biến byte. Nếu chương trình đầu vào tuân thủ đúng nguyên tắc sử dụng biến byte - biến bit, ta sẽ chuyển đổi các biến này thành một cấu trúc dữ liệu khác phù hợp hơn ở dạng ngôn ngữ cấp cao. Như đã phân tích từ trước ở chương 1, cấu trúc dữ liệu đó là union. Khi lập trình ở dạng mã assembly, lập trình viên không được phép xử lý các vùng nhớ trực tiếp mà phải thông qua thanh ghi, tuy nhiên, khi đã chuyển đổi về dạng ngôn ngữ cấp cao, ta có thể sử dụng trực tiếp tên biến trong các câu lệnh mà không cần trung gian qua thanh ghi nữa. Điều này giúp mã đầu ra dễ hiểu và trong sáng hơn.

Đặc điểm của giải pháp này là ta đã biết trước các bộ biến, nên ngay ở giai đoạn mã trung gian, ta sẽ thể hiện các biến bit dưới dạng truy xuất của union.

Như vậy, ta không cần phải xử lý lại các biến bit này ở giai đoạn sinh mã đầu ra nữa.

Như vậy, với giải pháp Kiểm tra kiểu này, ta đã có sẵn thông tin về bộ biến ngay từ đầu và chỉ cần kiểm tra xem người lập trình có tuân thủ đúng quy tắc không trước khi sinh ra mã ở ngôn ngữ cấp cao. Tuy nhiên, giải pháp còn nhiều hạn chế như phương pháp phân tích dữ liệu chưa đạt độ chính xác cao, cần người dùng phải chỉnh sửa lại khai báo theo mẫu quy định... Chính vì vậy, giai đoạn sau của luận văn đã phát triển một giải pháp mới có độ chính xác cao hơn và không cần chỉnh sửa mã đầu vào của người dùng, đó là giải pháp Suy luận kiểu. Giải pháp này sẽ được trình bày ở chương tiếp theo.

## 4 Suy luận kiểu - Type inference

Sau khi đã hiện thực giải pháp Kiểm tra kiểu, nhận thấy rằng giải pháp này còn nhiều hạn chế và có thể cải tiến thêm, luận văn đã phát triển thêm một giải pháp mới tốt hơn, không bắt buộc người dùng phải thay đổi code của mình theo mẫu khai báo, đó là Suy luận kiểu. Với giải pháp này, bằng các phép phân tích dữ liệu, trình dịch ngược sẽ tự động tìm ra được các bộ biến được sử dụng trong chương trình. Các bước của giải pháp Suy luận kiểu gồm có:

1. Dùng một phương pháp phân tích luồng dữ liệu để biết được giá trị của thanh ghi ACC tại mỗi điểm của chương trình.
2. Đi qua các câu lệnh sử dụng biến bit, ghi nhận giá trị hiện tại của thanh ghi ACC tại câu lệnh đó và đưa biến bit đó vào bộ biến phù hợp.

Như vậy, giải pháp này đã bỏ qua được bước đầu tiên, quy định mẫu khai báo, của giải pháp Suy luận kiểu. Ngoài ra, kỹ thuật phân tích Reaching definitions như đã trình bày ở chương trước có một số khuyết điểm, vì vậy, ở giải pháp này, chúng ta sẽ phân tích và tìm ra một kỹ thuật khác toàn diện hơn. Điều này sẽ được trình bày ở phần đầu tiên của chương, phần tiếp theo sẽ nói về cách quét các câu lệnh sử dụng biến bit và đưa thông tin vào một cấu trúc dữ liệu phù hợp.

### 4.1 Phân tích Constant propagation

Ở chương trước, phương pháp phân tích Reaching definitions đã được đề cập đến, tuy nhiên, phương pháp này chỉ áp dụng được cho trường hợp gán một biến byte trực tiếp cho thanh ghi ACC, vì vậy ta cần tìm một phương pháp khác phù hợp hơn. Phương pháp này cần phải tính toán được chính xác giá trị hiện có của thanh ghi ACC cho dù biểu thức bên phải của phép gán là gì. Ngoài ra, như một số trình bày ở trên, nếu thanh ghi ACC có thể có mang những giá trị khác nhau ở một câu lệnh sử dụng bit, thì mặc nhiên nguyên tắc bị vi phạm. Như vậy, ta chỉ xét tới các trường hợp giá trị ở thanh ghi ACC là một giá trị cố định, có thể tính toán được trước khi thực thi chương trình. Từ các yêu cầu trên, ta kết luận được phương pháp phân tích phù hợp nhất trong trường hợp này là Lan truyền hằng số - Constant propagation. Phương pháp này cho phép tính toán giá trị của các biến, cho biết được giá trị đó có phải là một hằng số tại một thời điểm của chương trình hay không.

Với phương pháp này, một biến có thể có ba giá trị sau:

- Top: Nghĩa là chưa biết được biến có giá trị gì.
- Hằng số: Nghĩa là đã xác định được giá trị của biến là một hằng số.
- Bottom: Nghĩa là biến có thể mang những giá trị khác nhau, tùy thuộc vào luồng chạy của chương trình.

Cái giá trị trên được minh hoạ ở hình ??

Giải thuật của Constant propagation gồm có các bước:

Từ giải thuật trên, ta thấy giá trị của các biến ở output chỉ có thể là hằng số hoặc bottom. Đây là một hàm, vì vậy nó sẽ luôn có điểm dừng chứ không bao giờ lặp mãi mãi.

Với phương pháp phân tích này, ta có thể giải quyết được vấn đề về phải của phép gán thanh ghi ACC không chỉ đơn giản là một biến byte. Ngoài ra, nó còn nhận biết được các biểu thức có giá trị giống nhau mặc dù hình thức bên ngoài khác nhau. (xem hình ??)

Một lưu ý là trong trường hợp này, ta không xét đến giá trị của thanh ghi ACC, mà ta chỉ xét đến giá trị của địa chỉ vùng nhớ thanh ghi ACC đang lưu giữ. Như vậy, chỉ có các câu lệnh dạng MOV ACC, [biểu thức] sẽ được xét đến. Khi gặp câu lệnh gán có dạng MOV ACC, #[biểu thức] thì đoạn mã phân tích sẽ xem như giá trị của ACC không phải là hằng số (bottom).

## 4.2 Quét các câu lệnh sử dụng biến bit

Sau khi đã biết được giá trị địa chỉ vùng nhớ mà thanh ghi ACC đang nắm giữ, ta sẽ chuyển sang bước quét các câu lệnh sử dụng biến bit. Ở bước này, ta vẫn sẽ sử dụng lại cấu trúc dữ liệu UnionDefine đã giới thiệu ở chương trước. Tuy nhiên, sau quá trình phân tích, ta chỉ biết được giá trị, chứ không biết được biến byte nào đang lưu trữ giá trị đó. Và để tăng tốc độ chương trình, ta sẽ lưu giá trị đó vào UnionDefine chứ không lưu biến byte, chỉ khi kết thúc quá trình quét dữ liệu, ta mới truy cập vào bảng lưu trữ các câu lệnh #DEFINE, so sánh và tìm ra được biến byte nào tương ứng với các giá trị đó. Như vậy, cần có thêm một trường để lưu giá trị trực tiếp trong UnionDefine.

Quá trình quét câu lệnh và thu thập dữ liệu của giải pháp Suy luận kiểu cũng tương tự như giải pháp Kiểm tra kiểu, nhưng thay vì ghi nhận biến bit và biến byte rồi kiểm tra chúng có cùng một bộ như khai báo hay không, ta sẽ kiểm tra trước hai biến đó có vi phạm nguyên tắc sử dụng hay không (ví dụ như biến bit đó đã thuộc một bộ khác trước đó, hoặc ở bit vị trí đó của bộ biến của biến byte đã ghi nhận một biến bite khác), nếu không thì ta sẽ ghi nhận mối quan hệ này vào danh sách UnionDefine đang lưu trữ.

Sau khi đã quét hết các câu lệnh ở các procedure, trước khi chuyển đổi các UnionDefine thành các khai báo ở ngôn ngữ C và thêm vào danh sách các biến toàn cục của program như giải pháp trước, ta phải thêm vào một bước chuyển đổi từ giá trị thành biến byte đại diện cho giá trị đó. Điều này có thể được thực hiện bằng cách chạy vòng lặp qua bảng lưu trữ các câu lệnh #DEFINE đã được thiết lập từ quá trình parse mã đầu vào.

Nếu như có một giá trị nào đó chưa được khai báo ở câu lệnh #DEFINE, ta sẽ đặt tên mới cho biến byte đó là LOCATION\_[giá trị].

Bước cuối cùng cũng tương tự giải pháp trước, đó là thay thế. Ngoài việc bỏ các câu lệnh gán thanh ghi ACC, thay thế biến ACC bằng các biến byte tương ứng, ta còn có thêm một bước là thay thế các biến bit bằng các truy xuất tới union. Ở giải pháp trước, vì ta đã biết ngay từ đầu biến bit nào thuộc chung

bộ với biến byte nào, nên ở giai đoạn giải mã, khi gặp một biến bit ta đã thay thế nó bằng truy xuất tới union. Còn ở giải pháp này, ta không biết được các bộ biến, nên ở giai đoạn giải mã ta sẽ xem các biến bit như một biến độc lập. Sau khi đã tìm ra được các bộ biến, ta mới thực hiện thay thế chúng.

Ngoài thay thế các biến bit, đoạn mã này còn quét các vị trí người dùng gọi trực tiếp một bit của thanh ghi ACC, ví dụ như ACC.1. Với trường hợp này, đoạn mã sẽ tìm biến bit ở vị trí số một của biến byte tương ứng với giá trị thanh ghi ACC, và thay thế đoạn mã truy xuất trực tiếp đó bằng tên biến bit.

Như vậy, giải pháp này đã giải quyết phần lớn các vấn đề đặt ra của bài toán.

## Tài liệu