CHƯƠNG 6
CẤP HỢP NGỮ

Cấp hợp ngữ khác với các cấp máy trước về cách thức thực thi. Cấp này được thực thi bởi trình biên dịch (compiler) chứ không phải trình phiên dịch (interpreter).

Trình biên dịch biến đổi chường trình của người sử dụng viết (chương trình nguồn – source promgram) thành chương trình đích. Chương trình đích còn được gọi là chương trình đối tượng (object program) hay mô đun đối tượng (object module). Trong biên dịch có 2 bước phân biệt:

1. Biên dịch - tạo ra chương trình đích từ chương trình nguồn.
2. Thực thi chương trình vừa được tao ra.

Bước 2 chỉ được bắt đầu khi bước 1 hoàn tất. Trong khi thực thi chương trình đích có 3 cấp máy được hiện diện: Cấp vi chương trình, cấp máy qui ước, cấp máy hệ điều hành. Kết quả là 3 chương trình: chương trình đối tượng (chương trình đích), hệ điều hành và vi chương trình tồn tại trong bộ nhớ của máy tính vào thời gian chạy chương trình.

1. Hợp ngữ là gì

Hợp ngữ là một ngôn ngữ mỗi câu lệnh của nó tương ứng với một lệnh máy. Tuy nhiên, chương trình viết bằng hợp ngữ dễ hiểu, dễ viết hơn viết bằng ngôn ngữ máy rất nhiều. Hợp ngữ sử dụng các từ viết tắt tiếng Anh để xây dựng các câu lệnh thay vì dùng các chữ số khó nhớ.

Chương trình viết bằng hợp ngữ có thể làm đực tất cả những gì mà chương trình viết bằng ngôn ngữ máy có thể làm, điều này thì chương trình viết bằng ngôn ngữ cấp cao không làm được.

Chương trình viết bằng hợp ngữ chỉ có thể chạy trên một họ máy, ngược lại chương trình viết bằng ngôn ngữ cấp cao có thể chạy trên nhiều họ máy.

1.1 Dạng lệnh của hợp ngữ

Mỗi lệnh của hợp ngữ chuẩn bao gồm 4 phần (trường), Ví dụ:

1. Trường nhãn: được sử dụng để khai báo nhãn, tên thủ tục, tên biến giúp cho các lệnh khác có thể tham khảo tới thay cho các địa chỉ bộ nhớ.

2. Trường thao tác: có thể chứa một từ viết tắt của mã lệnh thực hiện một thao tác nào đó (MOV, ADD,...) hay chứa một giả chỉ thị nếu lệnh phục vụ cho trình hợp dịch.

3. Trường toán hạng: được sử dụng để xác định các địa chỉ, các thanh ghi mà lệnh dùng làm các toán hạng. Trường toán hạng của lệnh nhẩy cho biết nhẩy tới đâu. Trường toán hạng của giả chỉ thị tuỳ thuộc vào giả chỉ thị (ví dụ: .MODEL small) cho biết dùng bao nhiêu không gian bộ nhớ.

4. Trường chú thích: được sử dụng để người lập trình ghi lời chú thích về cách làm việc của chương trình nhằm giúp người người đọc chương trình dễ hiểu hơn.

Label	Operation	Operand	Comment
DATA:	MOV	CX, AX	;copy AX vào CX

Hình 6.1: Khuôn dạng lệnh của hợp ngữ

1.2 So sánh hợp ngữ với ngôn ngữ bậc cao

Trong lịch sử phát triển máy tính, hợp ngữ đã được sử dụng trong nhiều năm do tính hiệu quả của nó (tiết kiệm tài nguyên). Tuy nhiên, do sự phát triển nhanh chóng của công nghệ phần cứng cũng như công nghệ phần mềm, ngày nay hợp ngữ không còn được sử dụng nhiều nữa. Người ta đã làm một cuộc thử nghiệm cho 2 hệ thống cùng chạy một chương trình, một sử dụng ngôn ngữ bậc cao và một dùng hợp ngữ. Kết quả là giá thành cho hệ thống dùng hợp ngữ lớn hơn gấp 5 lần do tốn công lao động.

Tuy nhiên, chương trình viết bằng hợp ngữ có tính hiệu quả hơn vì một lệnh viết bằng ngôn ngữ bậc cao tương đương với hàng chục lệnh hợp ngữ. Giá thành dùng hợp ngữ cao hơn vì hợp ngữ khó sử dụng hơn ngôn ngữ bậc cao.

Ngày nay một số trường hợp vẫn phải dùng hợp ngữ với lí do hạn chế tài nguyên về bộ xử lí và bộ nhớ. Một lý do khác nữa mà người ta vẫn dùng hợp ngữ là sự hiểu biết về hợp ngữ rất cần thiết cho việc hiểu cách làm việc của trình biên dịch.

Các nghiên cứu cho thấy trong đa số các chương trình, thường có một phần nhỏ của chương trình nhưng lại chiếm tới 90% thời gian thực thi chương trình. Ví dụ, trong trình biên dịch việc tìm kiếm bảng ký hiệu chiếm phần lớn thời gian so với phần còn lại của trình biên dịch. Vì vậy, để tăng tính hiệu quả mà vẫn giảm giá thành người ta kết hợp dùng cả hợp ngữ và ngôn ngữ bậc cao cho một chương trình. Phần lớn chương trình được viết bằng ngôn ngữ bậc cao, phần nhỏ nhưng chạy mất nhiều thời gian được viết bằng hợp ngữ.

1.3 Tiến trình hợp dịch

Một chương trình hợp ngữ bao gồm một chuỗi các câu lệnh, nên trình dịch hợp ngữ phải đọc từng câu lệnh, dịch thành ngôn ngữ máy và sau đó xuất ngôn ngữ máy đã được tạo ra lên một file cùng với phần danh sách tương ứng nếu có lên một file khác. Tiến trình này sẽ được lặp lại cho tới khi toàn bô chương trình được dịch. Cách làm việc trên không phải lúc nào cũng thuận lợi. Ví dụ, trường hợp câu lệnh đầu tiên là một chỉ thị nhẩy tới L. Trình dịch hợp ngữ không thể dịch câu lệnh này cho tới khi biết được địa chỉ của L. Câu lệnh L có thể ở gần cuối chương trình, do vậy trình dịch hợp ngữ không thể tìm địa chỉ của L mà không đọc trước hầu như toàn bộ chương trình. Khó khăn này được gọi là vấn đề tham chiếu trước. Bởi vì kkí hiệu L được sử dụng trước khi được định nghĩa (tham chiếu một kí hiệu mà kí hiệu này sẽ được định nghĩa sau).

Vấn đề "tham chiếu trước" có thể được giải quyết bằng 2 cách:

Cách 1: chương trình nguồn sẽ được trình dịch hợp ngữ đọc 2 lần. Mỗi lần đọc chương trình nguồn được gọi là một bước, một trình dịch đọc chương trình nguồn 2 lần được gọi là trình dịch 2 bước. Bước 1, các kí hiệu kể cả các nhãn của các câu lệnh được cất vào một bảng. Sang bước 2 tất cả các kí hiệu đã được biết rõ địa chỉ và như vậy không gặp phải vấn đề tham chiếu trước nữa. Như vậy lần lượt các câu lệnh được đọc, dịch và xuất ra file. Cách này đòi hỏi phải đọc chương trình nguồn 2 lần nhưng không phức tạp.

Cách 2: chỉ đọc chương trình nguồn một lần. Mỗi khi gặp một câu lệnh không thể dịch được do tham chiếu trước, sẽ không có lệnh mã máy được xuất ra file mà thay vào đó một điểm nhập được tạo ra trong một bảng để cho biết câu lệnh có tham chiếu trước chưa được dịch. Ở cuối tiến trình dịch, tất cả các câu lệnh chưa được dịch (ở trong bảng) sẽ được dịch.

Trong cách này, theo sau tiến trình dịch là một tiến trình nạp, chương trình nạp sẽ đặt các kết quả xuất vào đúng trật tự của chúng. Phương pháp này có nhược điểm là nếu chương trình có nhiều câu lệnh tham chiếu trước thì bảng chứa các câu lệnh để lại dịch sau chiếm mất nhiều bộ nhớ. Ngoài ra, trình dịch hợp ngữ một bước có độ phức tạp hơn so với trình dịch hợp ngữ 2 bước. Chính vì vậy, đa số các trình dịch hợp ngữ là 2 bước.

1.3.1 Bước 1

Chức năng chính của bước 1 là xây dựng bảng ký hiệu (symbol table) chứa giá trị của tất cả các ký hiệu. Một ký hiệu là một nhãn hoặc một giá trị được gán một tên ký hiệu bằng một giả lệnh. Ví dụ: bufsize equ 100.

Trong tiến trình dịch, trình dịch hợp ngữ phải biết địa chỉ của lệnh

đang được dịch, trình dịch hợp ngữ duy trì một biến trong thời gian dịch gọi là bộ đếm vị trí lệnh (instruction location counter – ILC). ILC = 0 lúc bắt đầu bước 1 và được tăng dần theo chiều dài của các lệnh được xử lí trong tiến trình của bước 1.Ví dụ:

Nhãn	Mã	Toán hạng	Chú thích	Chiều dài lệnh	ILC trước phát biểu
...					
HANOI1	MOV	EAX, I	; EAX = I	5	100
	MOV	EBX, J	; EBX = J	6	105
HANOI2	MOV	ECX, K	; ECX = K	6	111
	IMUL	EAX, EAX	; EAX = I*I	2	117
	IMUL	EBX, EBX	; EBX = J*J	3	119
	IMUL	ECX, ECX	; ECX = K*K	3	122
HANOI3	ADD	EAX, EBX	; EAX = I*I+J*J	2	125
	ADD	EAX, ECX	; EAX=I*I+J*J+K*K	2	127
HANOI4	MOV	N, EAX	; N=I*I+J*J+K*k	5	129
	JMP	DONE	; nhẩy tới DONE	5	134

Bước 1 của trình dịch hợp ngữ sử dụng ít nhất 2 bảng: bảng kí hiệu và bảng mã phép toán. Mỗi điểm vào của bảng kí hiệu chứa: ký hiệu hoặc một con trỏ trỏ tới ký hiệu, giá trị và những thông tin khác nếu có. Ví dụ:

Ký hiệu	Giá trị	Thông tin khác
HANOI1	100	
HANOI2	111	
HANOI3	125	
HANOI4	129	
...		

Hình 6.2: Bảng ký hiệu cho cho chương trình

Bảng mã phép toán chứa ít nhất một điểm nhập cho mỗi mã lệnh. Mỗi điểm nhập chứa: từ gợi nhớ của mã lênh, 2 toán hạng, mã 16 bit của mã lệnh, chiều dài lệnh và lớp lệnh. Ví dụ:

Mã lệnh (từ gợi nhớ)	Toán hạng 1	Toán hạng 2	Mã hexa của mã lệnh	Chiều dài lệnh	Lớp lệnh
AAA	---	---	37	1	6
ADD	EAX	IMMED32	05	5	4
ADD	REG	REG	01	2	19
ADD	EAX	IMMED32	25	5	4
ADD	REG	REG	21	2	19
...					

Hình 6.3: Một trích đoạn từ bảng mã lệnh cho trình dịch hợp ngữ của 80386

Ví dụ, xét mã lệnh ADD. Nếu lệnh ADD chứa EAX là toán hạng 1 và một hằng số 32 bit (IMMED32) là toán hạng 2, mã hexa là 05h và chiều dài lệnh 5 byte. Nếu lệnh ADD dùng 2 thanh ghi làm toán hạng thì chiều dài lệnh 2 byte và mã hexa là 01h. Lớp lệnh sẽ được cung cấp cho tất cả các kết hợp mã lệnh – toán hạng theo những qui luật giống nhau và được xử lí theo cùng một cách (ví dụ, lệnh ADD có 2 toán hạng là thanh ghi thuộc lớp 19). Trên thực tế lớp lệnh chỉ rõ một thủ tục trong trình dịch hợp ngữ, thủ tục này được gọi để xử lí tất cả các lệnh thuộc loại đã cho.

1.3.2 Bước 2

Bước 2 có nhiệm vụ tạo ra chương trình đối tượng (object program) và có thể in danh sách hợp dịch. Ngoài ra bước 2 còn phải xuất thông tin cần thiết cho trình liên kết (linker) để kết nối những thủ tục đã được hợp dịch ở những thời điểm khác nhau.

Thủ tục cho mỗi lớp của trình hợp dịch cho biết có bao nhiêu toán hạng mà lớp đó có thể có và gọi một thủ tục có tên: EvaluateExpression một số lần thích hợp. Thủ tục này có nhiệm vụ đổi biểu thức ký hiệu thành số nhị phân. Một khi đã biết được giá trị bằng số của mã lệnh và các toán hạng, lệnh hoàn toàn có thể được hợp dịch. Sau đó lệnh đã được dịch được đưa vào một bộ đệm xuất và được ghi lên đĩa khi bộ đệm đầy. Câu lệnh mã nguồn và câu lệnh mã đối tượng được tạo ra từ câu lệnh mã nguồn hoặc

được in hoặc được đưa vào bộ đệm để in sau. Sau khi ILC được điều chỉnh, câu lệnh kế tiếp sẽ được tìm nạp.

Ở trên, chúng ta giả thiết rằng chương trình nguồn không có bất kỳ lỗi nào. Trong thực tế, một chương trình nguồn thường mắc lỗi do đánh máy, sai cú pháp,... như:

+ Sử dụng ký hiệu nhưng không định nghĩa.

+ Định nghĩa một ký hiệu nhiều hơn một lần.

+ Tên trong trường mã lệnh không hợp lệ.

+ Mã lệnh không được cung cấp đủ toán hạng.

+ Mã lệnh được cung cấp quá nhiều toán hạng.

+ Dùng các chữ số không phù hợp với cơ số.

+ Dùng thanh ghi không hợp lệ (ví dụ, lệnh nhẩy tới một thanh ghi).

+ Thiếu câu lệnh END.

Lỗi do dùng ký hiệu chưa định nghĩa thường do lỗi đánh máy gây ra, vì vậy một trình hợp dịch thông minh có thể đoán ký hiệu nào trong số những ký hiệu đã định nghĩa giống với ký hiệu đang gặp lỗi nhất và thay vào đó. Một điều thường làm đối với các trình hợp dịch là khi gặp một câu lệnh sai thì in ra một thông báo lỗi và thử tiếp tục hợp dịch.

1.3.3 Bảng ký hiệu

Trong bước 1, trình hợp dịch tích luỹ thông tin về các ký hiệu và các giá trị của chúng, cất chúng vào trong bảng ký hiệu để sử dụng trong bước 2. Có nhiều phương pháp có thể được sử dụng để tổ chức bảng ký hiệu này.

Phương pháp đơn giản nhất là bảng ký hiệu bao gồm một dãy các cặp phần tử, phầnt tử thứ nhất là ký hiệu và phần tử thứ hai là giá trị. Khi cần tìm kiếm một ký hiệu, quá trình tìm kiếm được thực hiện một cách tuyến tính cho tới khi tìm thấy. Phương pháp này dễ thực hiện nhưng tốc độ thực thi chậm bởi vì tính trung bình mỗi lần phải tìm kiếm trên một nửa danh sách của bảng.

Một phương pháp khác là tổ chức bảng ký hiệu theo một dãy được xếp thứ tự trên các ký hiệu và dùng thuật toán tìm kiếm nhị phân để tìm kiếm một ký hiệu khi cần thiết. Với phương pháp này trung bình để tìm một ký hiệu của bảng có n ký hiệu chỉ mất $\log_2 n$ lần thử.

1.4 Macro

Khi viết chương trình hợp ngữ, có một số đoạn mã lệnh cần phải lặp đi lặp lại nhiều lần trong chương trình. Để tránh sự lặp lại đó, có một cách người ta thường làm là đưa đoạn mã lệnh đó vào trong một thủ tục, và gọi thủ tục đó mỗi khi cần lặp lại. Tuy nhiên, cách này có nhược điểm là với một nhóm lệnh ngắn mà phải gọi thường xuyên thì chi phí cho việc gọi thủ tục sẽ làm giảm tốc độ thực thi chương trình.

Có một cách hiệu quả hơn trong trường hợp nhóm mã lệnh cần lặp lại ngắn hoặc không thích hợp để viết thành thủ tục đó là dùng macro.

1.4.1 Định nghĩa, gọi và mở rộng macro

Định nghĩa macro là nhóm lệnh được gán cho một tên gọi ở đầu nhóm. Sau khi một macro đã được định nghĩa, người lập trình có thể viết tên macro thay vì viết lại nhóm lệnh. Thực tế macro là sự viết tắt cho một đoạn chương trình. Ví dụ:

```
PUSH_ALL MACRO
        PUSH AX
        PUSH BX
        PUSH CX
        PUSH DX
ENDM
```

Tổng quát, một định nghĩa macro bao gồm 3 phần:

1. Tiêu đề macro: cho biết tên của macro được định nghĩa.

2. Thân macro: nhóm mã lệnh cần lặp lại.

3. Một giả lệnh đánh dấu điểm kết thúc của định nghĩa macro.

Để gọi macro trong chương trình hoặc trong thủ tục, người lập trình chỉ cần viết tên của macro.

Khi trình hợp dịch gặp một định nghĩa macro, nó cất macro vào một bảng định nghĩa macro để dùng sau này. Kể từ khi đó, mỗi khi gặp một dòng lệnh là tên macro, trình hợp dịch sẽ thay thế tên này bằng thân của macro. Dòng lệnh tên macro được gọi là "lời gọi macro", việc thay thế tên macro bằng phần thân của macro được gọi là "mở rộng macro".

Việc mở rộng macro xẩy ra trong trời gian hợp dịch chứ không phải trong thời gian thực thi chương trình. Như vậy mã đích của một chương trình dùng macro và không dùng macro là giống nhau.

Việc gọi macro và việc gọi thủ tục có sự khác nhau. Lời gọi macro báo cho trình dịch hợp ngữ thay thế tên macro bằng phần thân của macro. Lời gọi thủ tục là một lệnh máy được chèn vào chương trình đối tượng và sẽ được thực thi sau để gọi thủ tục. Hình 6.4 so sánh lời gọi macro với lời gọi thủ tục.

Hạng mục	Gọi macro	Gọi thủ tục
Lời gọi được thực hiện khi nào?	Trong khi hợp dịch	Trong khi thực thi chương trình đối tượng.
Thân (macro/thủ tục) được chèn vào chương trình đối tượng ở mỗi nơi tên xuất hiện không?	Có	Không
Lệnh gọi thủ tục có được chèn vào chương trình đối tượng và được thực thi sau không?	Không	Có
Cần phải có lệnh trở về để điều khiển sự trở về sau khi thực hiện lời gọi không?	Không	Có
Có bao nhiêu bản sao của thân (macro/thủ tục) xuất hiện trong chương trình đối tượng?	Một bản sao cho mỗi lần gọi.	Chỉ có duy nhất 1 bản sao trong chương trình đối tượng.

Hình 6.4: Sự khác nhau giữa lời gọi macro với lời gọi thủ tục

1.4.2 Macro với các tham số

Thường chương trình hợp ngữ chứa các đoạn chương trình gần giống nhau. Để xử lý những trường hợp như vậy, trình dịch hợp ngữ cho phép các định nghĩa macro có tham số hình thức và cho phép các lời gọi macro cung cấp các tham số thực. Ví dụ, hình 6.5 (a) trình bày 2 đoạn mã gần giống nhau trao đổi giá trị các biến P, Q và R, S. Khi macro được mở rộng, mỗi tham số hình thức xuất hiện trong phần thân của macro được tha thế bởi tham số thực tương ứng. Các tham số thực được đặt trong trường toán hạng của lời gọi macro.

```
MOV EAX,P
MOV EBX,Q
MOV Q,EAX
MOV P,EBX
...
MOV EAX,R
MOV EBX,S
MOV S,EAX
MOV R,EBX
```

a)

```
CHANGE MACRO P1,P2
        MOV EAX,P1
        MOV EBX,P2
        MOV P2,EAX
        MOV P1,EBX
ENDM
...
CHANGE P,Q
...
CHANGE R,S
```

b)

Hình 6.5: Macro có tham số

1.4.3 Thực hiện tiện ích macro trong trình dịch hợp ngữ

Để thực hiện một tiện ích macro, trình dịch hợp ngữ phải thực hiện 2 chức năng: cất các định nghĩa macro và mở rộng các lời gọi macro.

Trình dịch hợp ngữ phải duy trì một bảng chứa tất cả các tên macro, cùng với mỗi một tên macro là một con trỏ trỏ tới định nghĩa macro được lưu trữ để tìm được khi cần.

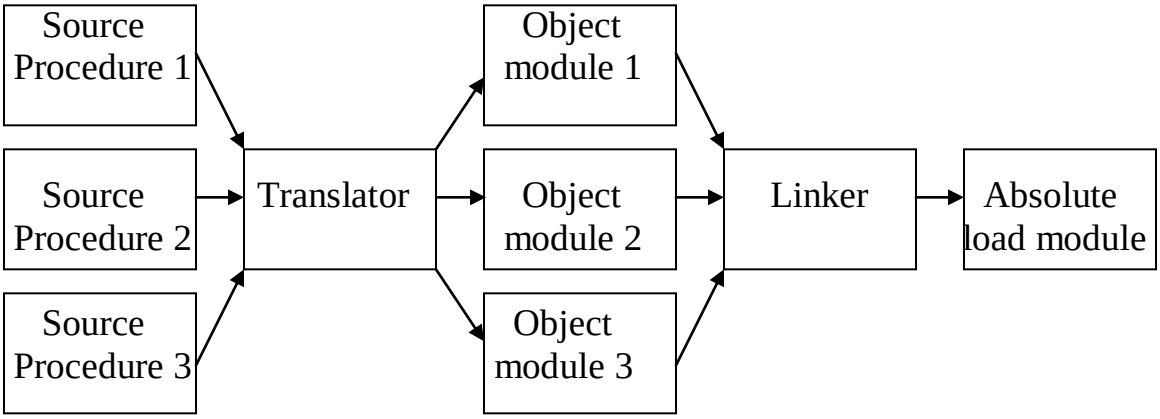
Khi gặp một định nghĩa macro, một điểm nhập của bảng được tạo ra chứa: tên macro, số các tham số hình thức và một con trỏ trỏ tới nơi lưu trữ định nghĩa macro (thân macro) trong bảng định nghĩa macro. Sau đó thân của macro được cất vào bảng định nghĩa macro. Trong bảng định nghĩa macro, thân macro được lưu giữ như là một chuỗi các ký tự. Để phân biệt biến hình thức có thể dùng thêm ký hiệu &, để thể hiện xuống dòng dùng thêm ký hiệu dấu chấm phẩy chẳng hạn. Ví dụ, thân macro CHANGE được lưu giữ trong bảng định nghĩa như sau:

MOV EAX,&P1; MOV EBX,&P2; MOV &P2,EAX; MOV &P1,EBX;

Trong giai đoạn bước 1 của hợp dịch, mỗi khi gặp một lời gọi macro, trình dịch hợp ngữ khởi động việc đọc phần thân của macro đã được cất giữ, các tham số hình thức ở trong phần thân của macro đã cất giữ được thay bằng các tham số thực được cung cấp trong lời gọi. Trình dịch hợp ngữ nhận ra các tham số hình thức thông qua ký hiệu &.

1.5 Liên kết và nạp

Thường các chương trình đều có nhiều hơn một thủ tục. Trình dịch hợp ngữ dịch các thủ tục và đặt lên đĩa. Trước khi chạy chương trình, tất cả các thủ tục đã dịch phải được liên kết với nhau một cách thích hợp (được thực hiện bởi trình liên kết - linker). Chương trình đã liên kết phải được đưa vào bộ nhớ ảo hay bộ nhớ chính nếu muốn thực thi (được thực hiện bởi trình nạp - loader).



Hình 6.6: Chương trình nguồn được dịch theo 2 bước tạo ra mô-đun nạp tuyệt đối

Việc dịch đầy đủ một chương trình nguồn để tạo ra một chương trình đích thi hành được phải trải qua 2 bước như minh hoạ trong hình 6.6.

1. Hợp dịch các thủ tục nguồn thành các mô-đun đối tượng.
2. Liên kết các mô-đun đối tượng để tạo ra mô-đun nạp tuyệt đối.

Bước 1 do trình dịch hợp ngữ thực hiện (ví dụ, trình dịch TASM.exe của Borland).

Bước 2 được thực hiện bởi trình liên kết (ví dụ, trình liên kết TLINK.exe của Borland).

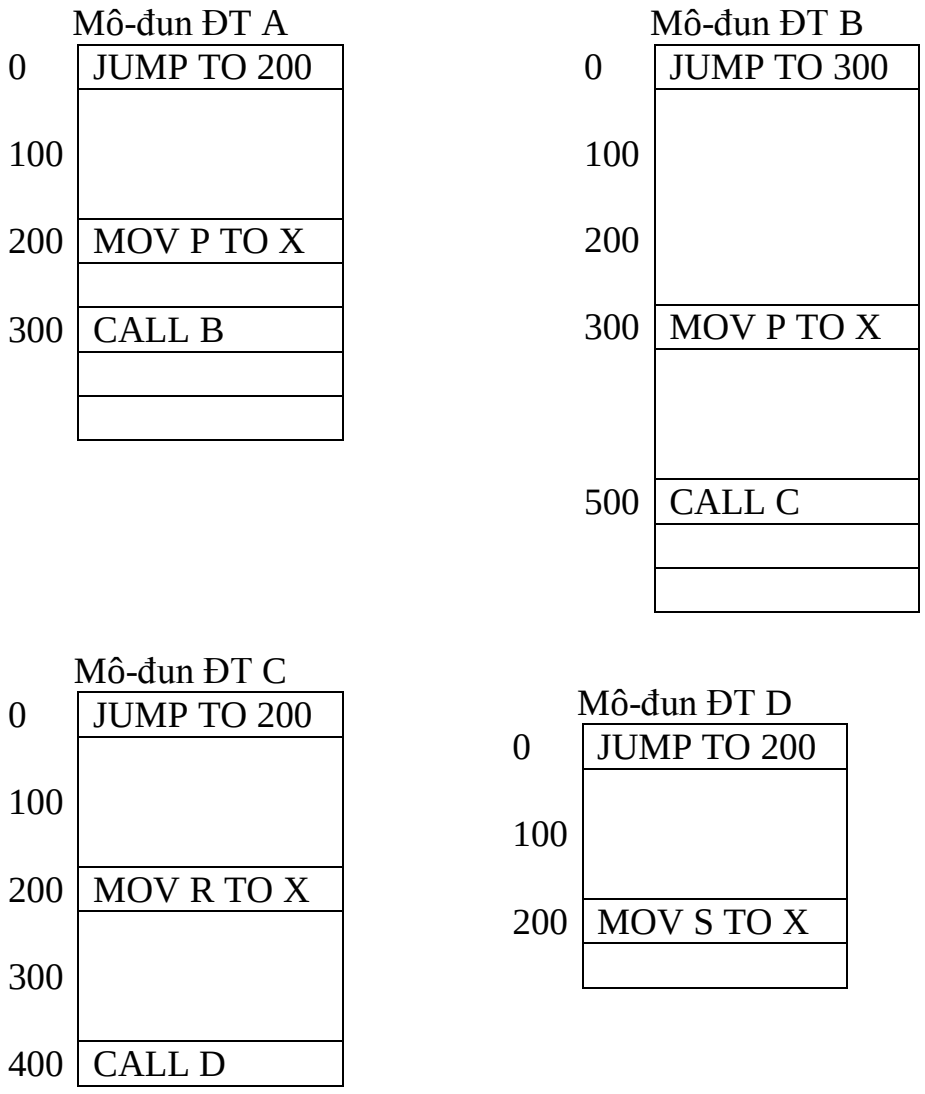
1.5.1 Các nhiệm vụ được thực hiện bởi trình liên kết

Ở bước một của tiến trình hợp dịch, mỗi mô-đun đối tượng đều có một không gian địa chỉ riêng và đều bắt đầu ở địa chỉ 0. Ví dụ, hình 6.7 trình bày 4 mô-đun đối tượng, mỗi mô-đun đều bắt đầu bằng một lệnh JUMP nhẩy tới lệnh MOV trong mô-đun.

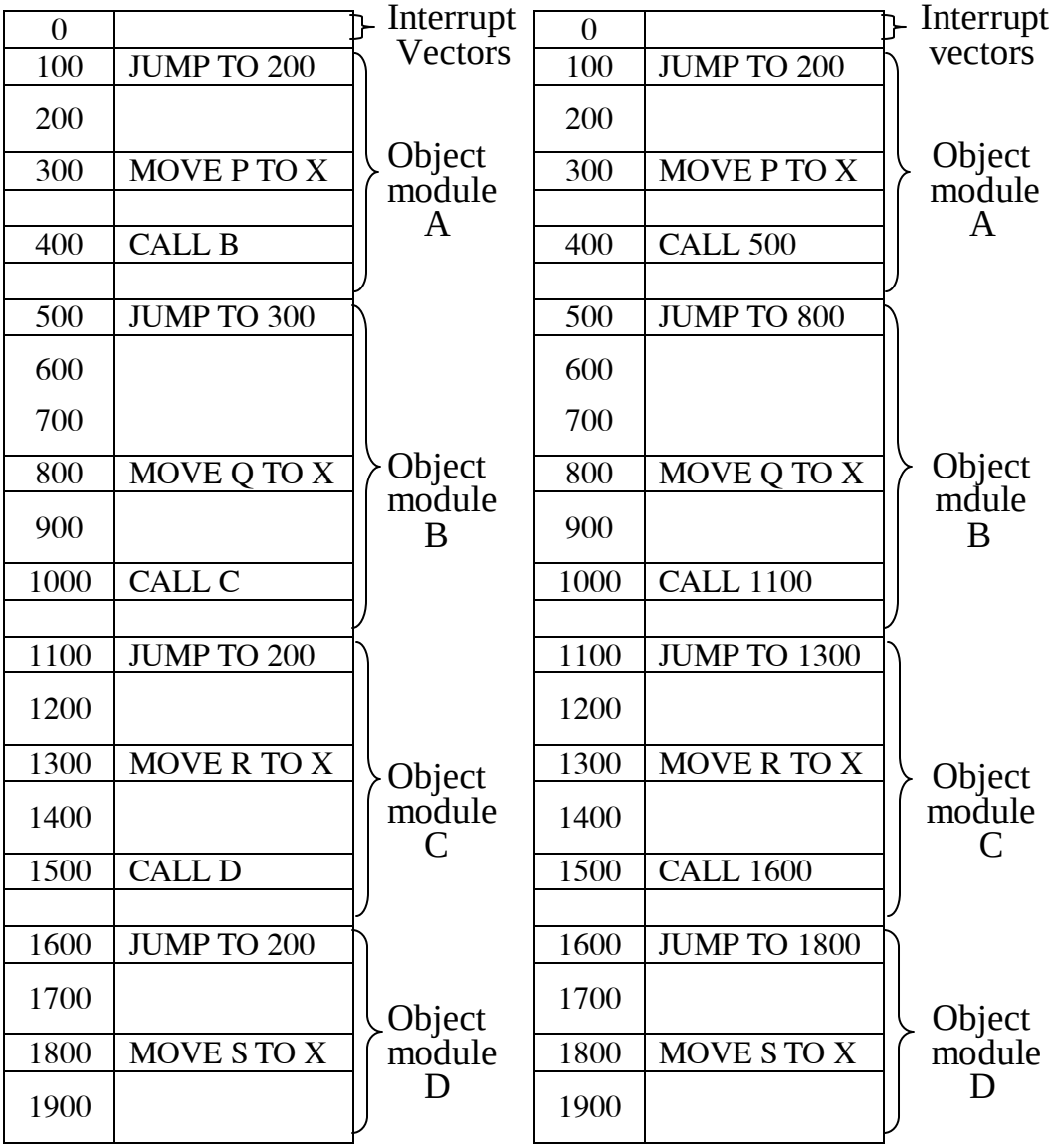
Để chạy chương trình, các mô-đun đối tượng phải được nạp vào bộ nhớ chính (hình 6.8 a). Một phần đầu của bộ nhớ chính dành cho các vecto ngắt và một số mục đích khác. Vì vậy các mô-đun đối tượng được nạp vào

từ địa chỉ 100 chẳng hạn. Tuy nhiên, chương trình đối tượng sẽ không thể thực thi bởi vì các địa chỉ tham chiếu bộ nhớ bây giờ đã bị sai lệch. Vấn đề này được gọi là "vấn đề tái định vị - relocation problem), xẩy ra do mỗi mô-đun đối tượng đều có một không gian địa chỉ riêng. Trên một máy tính bộ nhớ một chiều và tuyến tính, các mô-đun đối tượng phải được kết hợp thành một không gian địa chỉ du nhất.

Vấn đề thứ hai là các lệnh gọi thủ tục cũng không thể làm việc. Bởi vì trình hợp không thể xác định được địa chỉ của các mô-đun đối tượng. Vấn đề này gọi là "vấn đề tham chiếu ngoài – external reference). Cả 2 vấn đề trên sẽ được giải quyết bởi trình liên kết.



Hình 6.7: Mỗi mô-đun đối tượng có một không gian địa chỉ riêng



a) b)

Hình 6.8: a) Các mô-đun đối tượng được nạp trước khi liên kết

b) Các mô-đun đối tượng được nạp sau khi liên kết

Trình liên kết kết hợp các không gian địa chỉ riêng rẽ của các mô-đun đối tượng thành một địa chỉ tuyến tính theo các bước sau:

1. Xây dựng một bảng các mô-đun đối tượng bao gồm các trường: tên mô-đun, chiều dài mô-đun và địa chỉ bắt đầu.
2. Dựa vào bảng này, gán địa chỉ nạp cho các mô-đun đối tượng.

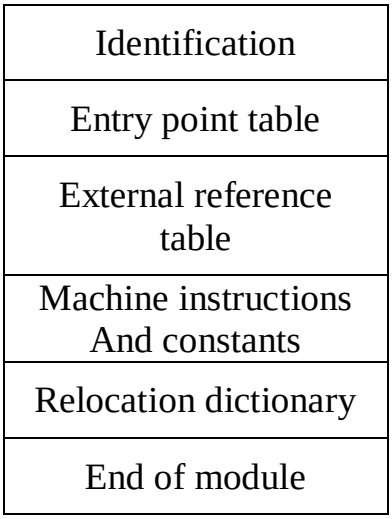
3. Tìm tất cả các lệnh chứa địa chỉ bộ nhớ và cộng từng địa chỉ bộ nhớ này với một hằng số tái định vị, hằng số này bằng với địa chỉ bắt đầu của mô-đun.
4. Tìm tất cả các tham chiếu tới những thủ tục khác và chèn địa chỉ của những những thủ tục này đúng chỗ.

Bảng mô-đun đối tượng xây dựng ở bước 1 cho các mô-đun ở hình 6.8 a như sau:

Mô-đun	Chiều dài	Địa chỉ bắt đầu
A	400	100
B	600	500
C	500	1100
D	300	1600

1.5.2 Cấu trúc của mô-đun đối tượng

Các mô-đun đối tượng có 6 phần được trình bày nhưng trong hình 6.9.



Hình 6.9: Cấu trúc của một mô-đun đối tượng

Phần 1 của mô-đun chứa tên mô-đun, các thông tin cần cho trình liên kết như: chiều dài của các phần khác nhau của mô-đun và ngày hợp dịch.

Phần 2 của mô-đun là một danh sách các ký hiệu đã định nghĩa trong mô-đun mà những mô-đun khác có thể tham chiếu cùng với các giá trị của chúng.

Phần 3 của mô-đun bao gồm các ký hiệu được sử dụng trong mô-đun nhưng được định nghĩa trong các mô-đun khác cùng với một danh sách các

lệnh nào sử dụng ký hiệu nào. Trình liên kết dùng danh sách sau để có thể chèn các địa chỉ đúng vào các lệnh sử dụng các ký hiệu ngoài.

Phần 4 của mô-đun chứa mã hợp dịch và các hằng số. Phần này sẽ được nạp và bộ nhớ để thực thi. 5 phần kia được trình liên kết sử dụng và sau đó bị loại bỏ trước khi bắt đầu thực thi.

Phần 5 của mô-đun là từ điển tái định vị, thông tin về những địa chỉ nào được tái định vị được cung cấp trong bảng này.

Phần 6 của mô-đun là một dấu hiệu kết thúc mô-đun.