TRƯỜNG ĐẠI HỌC BÁCH KHOA THÀNH PHỐ HỒ CHÍ MINH

KHOA KHOA HỌC & KĨ THUẬT MÁY TÍNH

KIẾN TRÚC MÁY TÍNH (C02008)

BÁO CÁO BÀI TẬP LỚN 02

NHÓM 02

ĐỀ 7: Xác định vị trí cuối cùng của chuỗi "Ten_nhom"
trong chuỗi "pString", chuỗi pString có N phần tử, N = 1000

Sinh viên thực hiện:

1. Nguyễn Minh Khôi – 1611657
2. Nguyễn Trọng Nghĩa – 1612212
3. Nguyễn Văn Tường - 1614028

Thành phố Hồ Chí Minh, tháng 12/2017

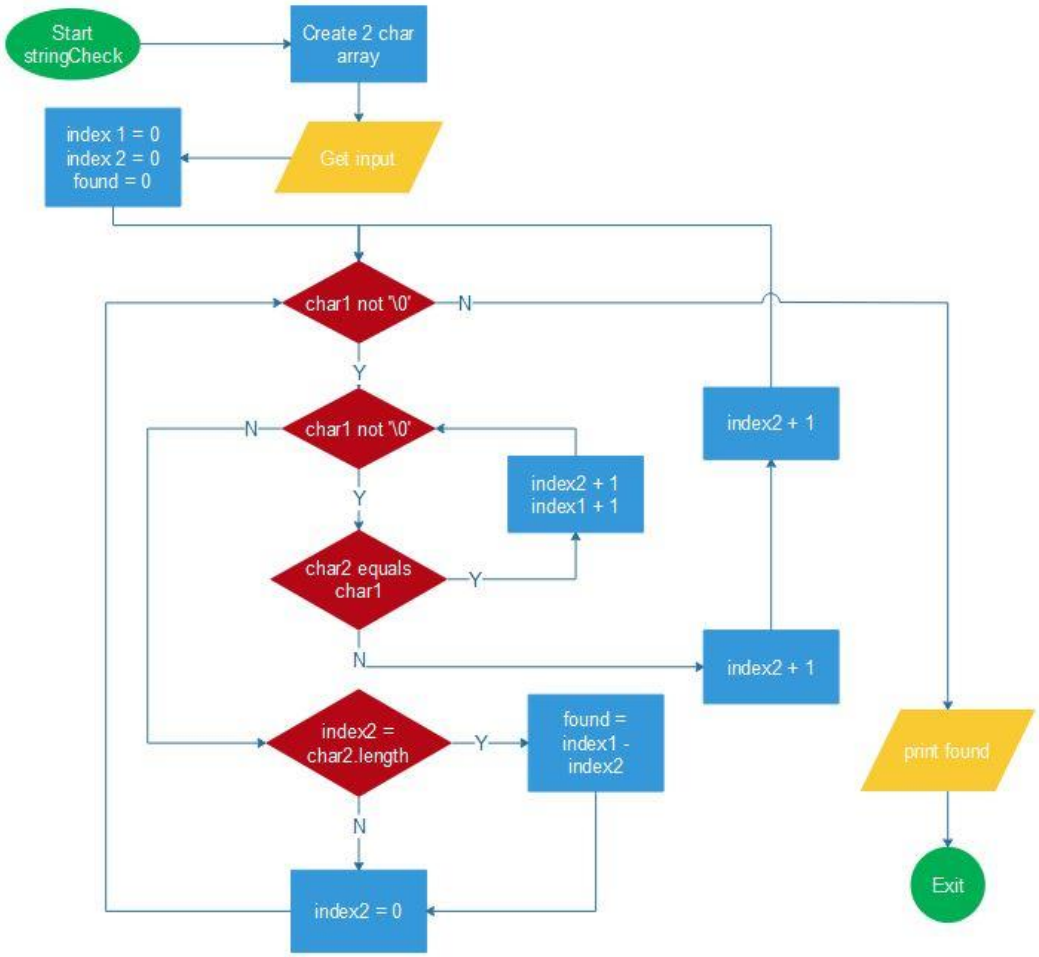
1. Thiết kế giải thuật

a. Ý tưởng

Gọi **mString** là chuỗi cần xác định nếu có tồn tại trong chuỗi **pString**. Trong khi phần tử đang xét của **pString** chưa phải là **'\0'**, ta duyệt tất cả các kí tự trong chuỗi **mString**, nếu phần tử của **pString** bằng với phần tử của **mString**, tăng chỉ số của **pString** và **mString** lên 1, nếu không, tăng chỉ số của **pString** lên 1 và thoát khỏi vòng lặp duyệt chuỗi **mString**. Nếu chỉ số của **mString** bằng với độ dài của chuỗi **mString**, lúc này chuỗi **mString** được tìm thấy, gán vị trí tìm thấy và gán chỉ số của chuỗi **mString** để chuẩn bị cho vòng lặp duyệt kí tự tiếp theo trong chuỗi **pString**. Sau khi kết thúc vòng lặp duyệt kí tự của **pString**, in ra màn hình giá trị tìm thấy.

Độ phức tạp của thuật toán trong trường hợp xấu nhất khi chuỗi **mString** nằm ở vị trí cuối cùng trong chuỗi **pString**, khi đó O(N) = N; trường hợp tốt nhất xảy ra khi chuỗi **mString** trùng với chuỗi **pString**, khi đó O(N) = N.

b. Flow-chart



c. Giải thuật viết bằng ngôn ngữ C (2_ktmt_11_code.c)

```c
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

#define NUM 1000

int main(int argc, char** argv) {
	// create base string to compare
	char* pString = (char*)malloc(NUM * sizeof(char));
	// create string to compare
	char* myString = (char*)malloc(NUM * sizeof(char));

	// ok, let's read from screen our 2 screen
	scanf("%s%s", pString, myString);

	// pIndex is the index of character in pString
	int pIndex = 0;
	// myIndex is the index of character in myString
	int mIndex = 0;
	// found will record the position of myString when matched pString
	int found = 0;
	while (pString[pIndex] != '\0') {
	// traversal the pString character, and each time, compare with myString
		while (myString[mIndex] != '\0') {
			// if myString character matched pString character
			// increase index of both
			// till the end of myString
			// if not reaching the end, which means unmatched, reset
myString index and increase pString index
			if (myString[mIndex] == pString[pIndex]) {
				mIndex++;
				pIndex++;
			}
			else {
				pIndex++;
				break;
			}
		}
		// after checking match, if mIndex equals to myString length
		// which means matched, record the position
		// 'cause we continuously searching, "found" at last will record the
final position those two matched
		if (mIndex == strlen(myString)) found = pIndex - mIndex;
		mIndex = 0;
	}

	// print the position of last found myString in pString
	printf("%d", found);
	return 0;
}
```

2. Tối ưu và tính toán thời gian thực thi

a. Phát hiện và giải quyết hazard

STT	Hazard	Chi tiết	Hướng giải quyết
0	Data hazard	Xảy ra khi quá trình xử lí ống không thể tiếp tục thực thi một giai đoạn nào đó cần phải chờ giai đoạn khác hoàn thành và cung cấp dữ liệu để tiến hành xử lí.	Chèn thêm các stall, sắp xếp lại lệnh.
1	Control hazard	Xảy ra khi bộ xử lí thực thi các lệnh rẽ nhánh có điều kiện và chưa xác định được lệnh tiếp theo sẽ là lệnh nào trong hai lệnh: lệnh ngay sau lệnh đang được thực thi và lệnh ở địa chỉ đích của lệnh rẽ nhánh.	Sử dụng phương pháp tiên đoán tĩnh, giả sử điều kiện rẽ nhánh luôn sai.

Bảng 1. Các rủi ro gặp phải trong file hợp ngữ và hướng giải quyết

Về thời gian thực thi, sử dụng công cụ *instruction counter* trên MARS 4.5 để tính toán số lệnh thực thi và áp dụng công thức: *clock cycles = 5 + (instructions − 1).* **Bộ test case sử dụng cho tính toán thời gian thực thi được đính kèm trong file báo cáo.**

b. Giải quyết data hazard bằng phương pháp sắp xếp lại lệnh

Giải quyết hazard bằng cách sắp xếp lại lệnh được thực hiện chi tiết trong file **7_ktmt_nhom11_MIPS_Reorder.asm.**

Bảng dưới minh họa một trường hợp giải quyết data hazard bằng rescheduling.

Mã nguồn gốc	Mã nguồn đã được giải quyết
75: lw $2,20($fp)	55: lw $v0,20($fp)
76: lw $3,32($fp)	56: lw $v1,32($fp)
77: addu $2,$3,$2	57: lw $a0,28($fp)
78: lbu $3,0($2)	58: addu $v0,$v1,$v0
79: lw $2,16($fp)	59: lbu $v1,0($v0)
80: lw $4,28($fp)	60: lw $v0,16($fp)
81: addu $2,$4,$2	61: addu $v0,$a0,$v0
82: lbu $2,0($2)	62: lbu $v0,0($v0)
83: bne $3,$2,.L4	63: bne $v1,$v0,.L4

Bảng 2. Ví dụ giải quyết data hazard bằng rescheduling

Sau khi giải quyết hazard, thời gian thực thi cụ thể cho từng trường hợp như sau.

	Trường hợp tốt nhất	Trường hợp xấu nhất	Trung bình
Số lệnh	36086	42818	39452
Số chu kì	36090	42822	39456

Bảng 3. Thời gian thực thi sau khi áp dụng phương pháp rescheduling

c. Giải quyết data hazard bằng phương pháp chèn stall

Giải quyết hazard bằng phương pháp chèn stall được thực hiện chi tiết trong file **6_ktmt_nhom11_MIPS_Hazard.asm.**

Bảng dưới mô tả cách giải quyết data hazard bằng phương chèn stall.

Mã nguồn gốc	Mã nguồn đã được giải quyết
132: lw $2,16($fp) 133: lw $3,28($fp) 134: addu $2,$3,$2 135: lbu $2,0($2)	131: lw $v0,16($fp) 132: lw $v1,28($fp) 133: nop → stall 134: addu $v0,$v1,$v0 135: lbu $v0,0($v0)

Bảng 4. Giải quyết data hazard bằng phương pháp chèn stall

d. Giải quyết control hazard bằng phương pháp tiên đoán tĩnh

Giải quyết hazard bằng phương pháp tiên đoán tĩnh được thực hiện chi tiết trong file **6_ktmt_nhom11_MIPS_Hazard.asm.**

Bảng dưới minh họa cách giải quyết control hazard bằng phương tiên đoán tĩnh.

Mã nguồn gốc	Mã nguồn đã được giải quyết
112: addu $v0,$v1,$v0 113: lbu $v0,0($v0) 114: bne $v0,$0,.L8 branch hazard	136: lbu $v0,0($v0) 137: nop 138: nop 139: bne 140: nop → chèn stall

Bảng 5. Giải quyết data hazard bằng phương pháp chèn stall

Sau khi giải quyết data hazard bằng phương pháp chèn stall và control hazard bằng phương pháp tiên đoán tĩnh, thời gian thực thi cụ thể cho từng trường hợp như sau:

	Trường hợp tốt nhất	Trường hợp xấu nhất	Trung bình
Số lệnh	40089	45917	43003
Số chu kì	40093	45921	43007

Bảng 6. Thời gian thực thi sau khi áp dụng hai phương pháp chèn stall và tiên đoán tĩnh

e. Chương trình tối ưu

Code hợp ngữ thu được sau khi dịch bằng cross compiler sử vùng nhớ để lưu các biến của chương trình thay vì các thanh ghi, dẫn tới sử dụng rất nhiều lệnh load và store để cập nhật giá trị của biến, làm giảm hiệu suất của chương trình. Bên cạnh đó, một số lệnh bị lặp lại liên tiếp không cần thiết đã làm tăng số lệnh thực thi, dẫn tới làm tăng thời gian xử lý.

Giải pháp tối ưu:

- Loại các lệnh load, store dư thừa và thay bằng các biến cố định.
- Loại bỏ các nhãn không cần thiết như L8, LC0, LC1.
- Không sử dụng thanh ghi $fp.

Code tối ưu được hiện thực chi tiết trong file **7_ktmt_11_MIPS_final.asm**
Bảng dưới minh họa một số trường hợp sử dụng tối ưu: (nhãn L6)

Mã nguồn gốc	Mã nguồn đã được giải quyết
75: lw $2,20($fp) 76: lw $3,32($fp) 77: addu $2,$3,$2 78: lbu $3,0($2) 79: lw $2,16($fp) 80: lw $4,28($fp) 81: addu $2,$4,$2 82: lbu $2,0($2) 83: bne $3,$2,.L4 84: nop 85: lw $2,20($fp) 86: addiu $2,$2,1 87: sw $2,20($fp) 88: lw $2,16($fp) 89: addiu $2,$2,1 90: sw $2,16($fp) 91: b .L3 92: nop	34: addu $v0,$s2,$t3 35: lbu $v1,0($v0) 36: addu $v0,$s1,$t2 37: lbu $v0,0($v0) 38: nop 39: nop 40: bne $v1,$v0,.L4 41: nop 42: addiu $t3,$t3,1 43: addiu $t2,$t2,1 44: b .L3 45: nop

Bảng 7. Tối ưu code sinh ra từ MIPS Codescape Cross Compiler

Sau khi tối ưu, thời gian thực thi cụ thể cho từng hợp như sau:

	Trường hợp tốt nhất	Trường hợp xấu nhất	Trung bình
Số lệnh	15051	22285	18668
Số chu kì	15055	22289	18672

Bảng 8. Thời gian thực thi sau khi tối ưu

ꕥ Hết ꕥ