Đại học Khoa học tự nhiên

Môn học: Hệ thống máy tính

BÁO CÁO ĐỒ ÁN CUỐI KỲ

Sinh viên:

Hùng Ngọc Phát – 19120615

Nguyễn Trọng Thái – 19120652

Nguyễn Hữu Nhật Tân – 19120647

Giảng viên:

Thái Hùng Văn

Mục lục

1 Giới thiệu

Phân công công việc:

- Nguyễn Trọng Thái (19120652) và Nguyễn Hữu Nhật Tân (19120647): làm câu 1. Chi tiết hơn ghi trong mục **Câu 1**.
- Hùng Ngọc Phát (19120615): Làm câu 2, tổng kết báo cáo và gõ LaTeX.

Ai làm câu nào thì cũng tự soạn luôn báo cáo cho phần đó.

Mã nguồn các chương trình và nội dung báo cáo này, ngoại trừ những phần được trích dẫn, là sản phẩm trí tuệ của các thành viên nhóm.

Phần lớn kiến thức liên quan đến học phần mà không được trích dẫn đến từ tài liệu môn Hệ thống máy tính của thầy Thái Hùng Văn [3].

Trong khi thực hiện đồ án này không thể tránh khỏi những sai sót. Nếu thầy có thắc mắc về sản phẩm của chúng em, xin thầy vui lòng liên hệ đại diện là: Hùng Ngọc Phát (19120615). Email: `19120615@student.hcmus.edu.vn`.

2 Câu 1

2.1 Giới thiệu

Chương trình ở câu 1 được viết bằng ngôn ngữ C++ theo chuẩn C++14 của Visual Studio.

Giả sử CPU đang chạy chương trình này sử dụng kiến trúc x86 nên dữ liệu nhị phân khi đọc hoặc ghi mặc định sẽ là little endian.

2.2 Danh sách các module

Chương trình được chia thành các file như sau:

- `main.cpp`: chứa hàm main và các hàm liên quan đến menu.
  Người thực hiện: Nguyễn Trọng Thái.
- `read.h`: chứa các hàm liên quan đến việc đọc các dữ liệu từ file nhị phân theo yêu cầu đề.
  Người thực hiện: Nguyễn Trọng Thái.
- `write.h`: chứa các hàm liên quan đến việc ghi dữ liệu xuống file nhị phân theo yêu cầu đề.
  Người thực hiện: Nguyễn Hữu Nhật Tân.

- `utils.h`: các hàm tiện ích.

  Người thực hiện: Nguyễn Hữu Nhật Tân.

Ngoài chức năng đọc/ghi chuỗi `UTF-16` (không thực hiện được), các chức năng còn lại theo yêu cầu đề đều hoạt động 100% với bộ test của tụi em, bao gồm cả đọc/ghi chuỗi `ASCII` thay cho `UTF-16`.

Vậy tổng thể chương trình này (Câu 1) đã hoàn thành được 80%.

2.3 Danh sách các hàm

Lưu ý: cả chương trình sẽ dùng chung một biến toàn cục để thực hiện việc đọc/ghi lên file nhị phân. File này sẽ được mở/đóng từ hàm main.

```
extern FILE* f = NULL;
```

2.3.1 Trong module `main.cpp`

- `int main()`: Hàm main.
- `void showWriteMenu()`: để hiện menu xử lí việc ghi dữ liệu.
- `void showReadMenu()`: để hiện menu xử lí nhập dữ liệu.

2.3.2 Trong module `utils.h`

- `static void reverseEndianess(uchar* arr, size_t n)`: Đảo ngược một mảng có kích thước `n` (để chuyển đổi giữa BE với LE).
- `static char chooseEndianess()`: để yêu cầu người dùng nhập vô là nhập LE hay BE. Trả về `'L'` hoặc `'B'` tương ứng với LE và BE.

  Các hàm này cần là ***`static`*** *vì file này có thể được include nhiều lần.*
- Ngoài ra trong file này cũng định nghĩa kiểu viết tắt `uchar` thay cho `unsigned char` là `typedef unsigned char uchar;`. Tụi em sử dụng kiểu `unsigned char` để đảm bảo các số biểu diễn dãy byte luôn dương.

2.3.3 Trong module `read.h`

- `void readBinary(uchar* buffer, size_t size)` : Để đọc dữ liệu có kích thước `n` từ file nhị phân lưu vào `buffer`.
- `void docSoNguyenQuaK()`: thực hiện chức năng đọc số nguyên quá K từ file nhị phân.
- `void docSoNguyenBu2()`: thực hiện chức năng đọc số nguyên bù 2 từ file nhị phân.
- Các hàm khác để thực hiện các chức năng còn lại theo yêu cầu đề bài.

2.3.4 Trong module `write.h`

Tương tự như trong `read.h`, nhưng các hàm này dùng để ghi xuống file nhị phân các dữ liệu theo yêu cầu đề bài.

2.4 Một số lưu ý về cơ chế hoạt động

- Để xử lí chuyển đổi giữa BE và LE, ta cần đọc các byte dữ liệu của biến cần chuyển và đưa nó vào một mảng kiểu `uchar[]`. Ví dụ như: biến số nguyên `int a = 0xaabbccdd` (giả sử 4 byte) sẽ được lưu vào một mảng `uchar buffer[4] = {0xdd, 0xcc, 0xbb, 0xaa}` (little endian). Để chuyển nó thành big endian, ta chỉ cần đảo ngược mảng đó lại.

  Ta cần sử dụng hàm `memcpy(dest_ptr, source_ptr, size)` của C để copy dữ liệu nhị phân từ 1 biến (nguyên, thực, ...) sang 1 mảng buffer như trên và ngược lại. Trong code thì em có dùng cú pháp `(uchar*)(&a)` để ép kiểu con trỏ vùng nhớ trỏ tới `a` (không cần biết kiểu dữ liệu của `a`) sang kiểu `uchar*` để chuyển nó thành một dãy các phần tử, mỗi phần tử 1 byte để có thể lưu trữ cho chính xác.[4]

  Vd: để copy dữ liệu nhị phân của 1 biến `double a = 2021.2022` (8 byte) thì ta có thể làm như sau:

```
double a = 2021.2022;
int size = 8;

uchar* buffer = new uchar[size];
memcpy(buffer, (uchar*)(&a), size);
...
delete[] buffer;
```

  Để làm ngược lại (chuyển dữ liệu từ byte array sang một biến nào đó), ta có thể làm như sau:

```
long a;
int size = 4;

uchar* buffer = {0x0f, 0x0d, 0x00, 0x01};
// Hoặc là fread(buffer, ...);
memcpy((uchar*)(&a), buffer, size);
```

```
...
delete[] buffer;
```

- Khi ghi hoặc đọc một dữ liệu bất kì (kiểu int, float, ...) trong code sẽ đều được chuyển thành một byte array `uchar*` trước khi ghi xuống hoặc sau khi đọc lên. Nếu người dùng muốn đọc/ghi dữ liệu BE thay cho LE cũng có thể chuyển đổi một cách dễ dàng bằng phương pháp này bằng cách đảo ngược byte array đó lại.

- Trong hàm *ghi* số nguyên bù 2 xuống file nhị phân, do C++ tự xử lí định dạng số nên ta chỉ cần đọc một số từ bàn phím (ko cần biết âm dương) và ghi nó xuống file là xong. Dữ liệu nhị phân trong file sẽ tự động là biểu diễn bù 2 của số nguyên đó (đã kiểm chứng qua thực nghiệm).

- Tuy nhiên, trong hàm *đọc* số nguyên bù 2 thì do kiểu dữ liệu của biến chứa em để là `long long` (8 byte, để có thể chứa được mọi số nguyên có thể trong C) nên nhiều khi số nguyên của mình chỉ có 4 byte thôi, khi đọc thì 4 byte còn lại sẽ bị thiếu. Nếu nó là số âm thì mình cần gán phần bị thiếu đó bằng `0xfff...`, còn nó là số dương thì mình cần gán nó bằng `0x000...`.

  Ví dụ giả sử số nguyên mà ta cần đọc là 4 byte, có dạng `0xffaabbcc` (số âm), nhưng biến để chứa nó là `long long` 8 byte. Khi đọc vào nó sẽ trở thành `0x00000000ffaabbcc` (số dương). Do đó ta cần xét bit dấu và chỉnh nó lại thành `0xffffffffffaabbcc` (số âm) mới đúng.
  Chi tiết hơn thầy có thể xem trong chương trình.

2.5 Chạy thử

Ở đây em sẽ lần lượt nhập các thông tin sau:

Dữ liệu	**Giá trị (dec)**	**Kích thước**	**LE/BE**
Số nguyên bù 2	-100	1 byte	LE
Số nguyên bù 2	2021	4 byte	BE
Số nguyên quá 100	-70	4 byte	LE
Chuỗi ASCII	he thong may tinh	17 kí tự	BE
Số thực chấm động	192.168	4 byte	LE

Nội dung file nhị phân sau khi ghi liên tiếp các thông tin trên (do gõ trên LaTex nên chèn hình hơi khó, em xin paste cho nhanh).

```
00000000: 9c00 0007 e51e 0000 0000 686e 6974 2079  .........hnit y
00000010: 616d 2067 6e6f 6874 2065 6802 2b40 43    am gnoht eh.+@C
```

Chúng ta có thể tính (bằng tay) được các offset của các thông tin ở bảng trên lần lượt là `0, 1, 5, 9, 27`. Tiếp theo ta sẽ thử chức năng đọc dữ liệu nhị phân từ các offset trên với định dạng tương ứng. Kết quả đọc được ở hình dưới đây:

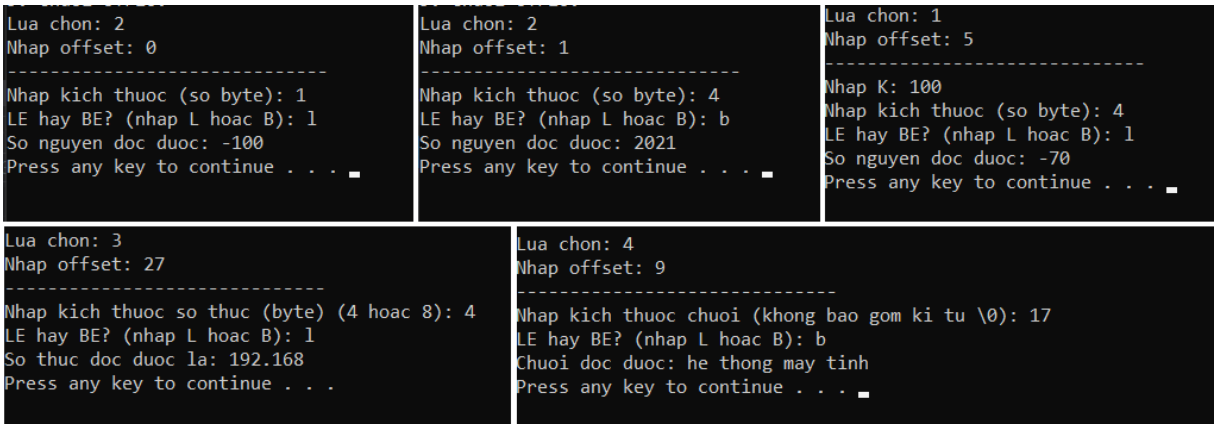

Figure 1: Kết quả khi đọc tại các offset trên của file nhị phân đã ghi ở bước trước

Ta có thể thấy được các dữ liệu đọc được hoàn toàn khớp với các dữ liệu ta đã nhập vào.

3 Câu 2

3.1 Giới thiệu

Chương trình ở câu 2 được viết bằng hợp ngữ `8086`, biên dịch bởi trình hợp dịch `emu8086 v4.08` và được test trên `DOSBox 0.74-3`.

Chương trình không sử dụng thư viện nào khác ngoài các ngắt `21h, 10h` và `15h` của DOS cũng như BIOS.

Nhắc lại chức năng của chương trình:

- Dọc trạng thái của 2 đèn NumLock (NL) và CapsLock (CL) [Hoàn thiện: 100%].
- Nếu đèn NL và CL đều sáng, in ra ngày giờ của máy tính dưới dạng đồng hồ *tự refresh sau mỗi 1 giây* [Hoàn thiện: 100%].
- Nếu chỉ có đèn NL sáng, tính tổng, hiệu, tích, thương của 2 số tự nhiên M và N nhập từ bàn phím sao cho $0 \leq M, N < 10$ [Hoàn thiện: 100%].
- Nếu chỉ có đèn CL sáng, vẽ một hình chữ nhật kích thước $M \times N$ bằng các dấu hoa thị * sao cho M, N nhập từ bàn phím và $0 \leq M, N < 10$ [Hoàn thiện: 100%].

3.2 Sơ lược về các 8086 instruction được sử dụng trong bài này[1]

Một instruction có thể có rất nhiều lựa chọn tham số khác nhau, nhưng trong phần này em sẽ chỉ nêu các lựa chọn có sử dụng trong đồ án này mà thôi.

3.2.1 Một số lệnh đơn giản

Bao gồm `MOV, LEA, INT, Jxxx, RET, ...`

3.2.2 `ADD` và `SUB`

Cú pháp: `ADD/SUB A, B`.

2 lệnh này nhận vào 2 tham số tổng quát là `A` và `B`, và thực hiện phép gán `A=A+B` hoặc `A=A-B`. Trong bài này chỉ sử dụng lệnh 2 lệnh đã nêu với `A` là một thanh ghi 8-bit.

3.2.3 `MUL`

Cú pháp: `MUL X`.

Nếu `X` là một byte, lệnh `MUL` sẽ thực hiện phép gán `AX=AL*X`.

Nếu `X` là một word, lệnh `MUL` sẽ thực hiện phép gán `(DX:AX)=AX*X`.

Ghi chú: kí hiệu ***`DX:AX`*** *nghĩa là một thanh ghi 32-bit "ảo" được tổ hợp từ 2 thanh ghi 16-bit* ***`DX`*** *và* ***`AX`*** *với* ***`DX`*** *là phần high và* ***`AX`*** *là phần low.*

Nghĩa là giá trị chứa trong thanh ghi `DX:AX` là `DX * 10h + AX` [5].

3.2.4 `DIV`

Cú pháp: `DIV X`.

Nếu `X` là một byte, lệnh `DIV` sẽ thực hiện phép toán `AX/X`, sau đó lưu thương số vào `AL` và dư số vào `AH`.

Nếu `X` là một byte, lệnh `DIV` sẽ thực hiện phép toán `(DX:AX)/X`, sau đó lưu thương số vào `AX` và dư số vào `DX`.

3.2.5 `PUSH` và `POP`

Cú pháp: `PUSH/POP X` với `X` là thanh ghi 16-bit.

Cặp lệnh này để thao tác với ngăn xếp (stack) được implement sẵn trong VXL 8086 [6]. Thường được gọi một cách thủ công để backup lại dữ liệu của các thanh ghi (sau hoặc trước một quá trình tính toán nào đó) trước khi ghi đè chúng để thực hiện các công việc khác.

Chúng ta cũng cần lưu ý không được làm corrupt ngăn xếp này (`PUSH` mà không `POP`, hoặc `POP` nhiều hơn `PUSH`, ...) vì nó cũng được sử dụng làm callstack cho các thủ tục. Khi đó các thủ tục sau khi kết thúc sẽ bị return về sai vị trí.

3.3 Sơ lược về các 8086 interrupt được sử dụng trong bài này[2]

Ngắt và hàm	Tác dụng	Tham số	Trả về
int 21h, AH=02h	In một kí tự ra màn hình	DL = Mã ASCII kí tự cần in	AL = DL
int 21h, AH=09h	In một chuỗi ra màn hình. Chuỗi phải kết thúc bằng dấu $	DS:DX = địa chỉ của chuỗi	Không
int 21h, AH=0Ch	Flush input buffer và gọi một hàm đọc khác	AL = STT của hàm đọc cần gọi	Tuỳ vào hàm đọc đã gọi
int 21h, AH=4Ch	Trả quyền điều khiển về cho hệ điều hành	Không	Không
int 21h, AH=2Ah	Lấy ngày hệ thống	Không	CX = năm (1980–2099), DH = tháng, DL = ngày, AL = thứ
int 21h, AH=2Ch	Lấy giờ hệ thống	Không	CH = giờ, CL = phút, DH = giây
int 10h, AH=02h	Đặt vị trí con trỏ	DH = hàng, DL = cột, BH = trang	Không
int 10h, AH=03h	Lấy vị trí con trỏ	BH = trang	DH = hàng, DL = cột
int 15h, AH=86h	Báo BIOS đợi một khoảng thời gian	CX:AX = thời gian bằng micro giây	Không

3.4 Chi tiết về các hàm và thủ tục được sử dụng

3.4.1 Danh sách các thủ tục và chức năng

Dưới đây là bảng tổng quát các thủ tục em đã viết thêm để có thể lập trình một cách dễ dàng hơn.

Tên thủ tục	Chức năng chính
main	Hàm main
printNumberAL	In một số có 1 hoặc 2 chữ số trong thanh ghi AL
printDX	In một chuỗi có địa chỉ trong DX
printRect	Thực hiện chức năng in một hình chữ nhật kích thước MxN
calculate2Num	Thực hiện chức năng tính các phép toán trên 2 số M và N
showClock	Thực hiện chức năng hiển thị đồng hồ

3.5 Hàm `main`

Hàm main thực hiện các chức năng sau:

- Nạp các biến cần thiết từ bộ nhớ vào data segment.
- Đọc và trích xuất các dữ liệu cần thiết từ *BIOS Data Area (BDA)* [7].
- Dựa vào các dữ liệu trích xuất được mà thực hiện các chức năng phù hợp theo yêu cầu đề bài.

Để đọc trạng thái các phím CapsLock (CL) và NumLock (NL), em đã đọc và trích xuất các thông tin ở BDA. Cụ thể như sau:

- Dữ liệu cần thiết để đọc trạng thái các phím CL và NL gọi là *Keyboard Flag Bytes*, gồm 4 byte.
- Chúng ta chỉ cần byte 0, nó chứa trạng thái của các phím NumLock, ScrollLock, CapsLock cũng như một số phím khác như Ctrl, Alt, ...
- Byte 0 của KBF nằm ở vị trí `40h:17h` trong bộ nhớ [7], nên ta cần thêm thanh ghi đoạn `ES` để chứa giá trị `40h` (trong code hợp ngữ).
- Trong byte 0, chúng ta chỉ cần 2 bit là bit số 5 (trạng thái NL) và bit số 6 (trạng thái CL).

```
40:17 byte Keyboard flag byte 0 (see KB FLAGS)

|7|6|5|4|3|2|1|0| keyboard flag byte 0
 | | | | | | | `--- right shift key depressed
 | | | | | | `---- left shift key depressed
 | | | | | `----- CTRL key depressed
 | | | | `------ ALT key depressed
 | | | `------- scroll-lock is active
 | | `-------- num-lock is active
 | `--------- caps-lock is active
 `---------- insert is active
```

Figure 2: Cấu trúc của ô nhớ số 17h tại offset 40h trong BDA [7]

- Để đọc 2 bit này, ta gọi lệnh `SHL` (dịch trái) 3 lần và đọc các bit bị "rơi" ra qua thanh ghi cờ `CF`.

3.6 Thủ tục `printDX`

Hàm này chức năng không có gì quá đặc biệt, chỉ rút ngắn 2 dòng lệnh in chuỗi thành còn một dòng

```
lea dx, msg
mov ah, 2
int 21h
; Chuyển thành
lea dx, msg
call printDX
```

3.7 Thủ tục `printNumberAL`

Vì input bị giới hạn lại là 2 số $m, n \in [0, 9] \subset \mathbf{N}$, nên kết quả nguyên dương lớn nhất có thể nhận được trong chương trình này là $m \times n = 81$. Do đó ta chỉ cần một hàm để in một số nguyên có 1 hoặc 2 chữ số ra màn hình. Hàm này hoạt động theo nguyên lý sau:

- Để chuyển giá trị của một chữ số bất kì `0..9` sang biểu diễn ASCII tương ứng của nó, ta cộng thêm nó với mã ASCII của chữ số `0` là `30h`.

- Nếu số đó chỉ có một chữ số (base case), ta cộng nó với `30h` và in giá trị nhận được ra màn hình.

- Nếu số đó có 2 chữ số, ta chia số đó cho 10. Khi đó ta thu được kết quả với dư số là chữ số thứ hai và thương số là chữ số thứ nhất. Sau đó, với mỗi chữ số nhận được, ta lại thực hiện tương tự như base case.

3.8 Thủ tục `printRect`

Hàm này để thực hiện chức năng vẽ hình chữ nhật có kích thước $M \times N$ với M, N nguyên dương, có một chữ số, được nhập từ bàn phím.

Cách thực hiện khá đơn giản, ta chỉ cần sử dụng 2 vòng lặp lồng nhau như sau (mã giả C, đã tối ưu để dễ chuyển qua hợp ngữ hơn):

```
int M, N; // 2 kích thước của hcn nhập từ bàn phím
scanf("%d %d", &M, &N);
int m = M, n = N; // 2 biến chạy
while (m > 0) {
    printf("\r\n");
    n = N;
    do {
        printf("*");
        n--;
    } while (n > 0);
    m--;
}
```

Khi chuyển qua hợp ngữ, ở các vị trí của từ khoá `while` ta chỉ cần thay bằng các cặp lệnh `cmp` và `jxx`. Cụ thể hơn thầy có thể xem source code của chương trình.

3.9 Thủ tục `calculate2Num`

Hàm này để thực hiện chức năng tính các giá trị tổng, hiệu, tích, thương của 2 số tự nhiên m, n thoả điều kiện trong yêu cầu đề bài.

- Với tổng $m + n$ và tích $m \times n$, ta lần lượt sử dụng các instruction `add` và `mul`. 2 công việc này khá đơn giản nên em sẽ không giải thích thêm nhiều.

- Với hiệu $m - n$ thì khó hơn một tí, đó là ta phải xét trường hợp $m < n$, khi đó hiệu là một số âm. Hàm `printNumberAL` mà em đã viết không handle được số âm

nên ta "lách luật" một chút, đó là tính và in giá trị $n - m > 0$ ra màn hình, sau đó in thêm dấu trừ ở đằng trước là được. Còn trong trường hợp $m > n$ thì hiệu $m - n > 0$ nên ta làm tương tự như với tổng.

- Với phép chia thì khá là phức tạp, vì kết quả không phải lúc nào cũng nguyên. Để biểu diễn kết quả của $m \div n$, em sẽ sử dụng kết quả của phép chia Euclide [8]:

$$\begin{aligned} & m = nq + r \\ \Leftrightarrow & \frac{m}{n} = q + \frac{r}{n} \end{aligned}$$

Ví dụ, khi $m = 9, n = 2$ thì $q = 4, r = 1$ nên kết quả được hiển thị ra màn hình sẽ là `4+1/2`. Còn khi $r = 0$ thì ta không cần hiển thị phần từ dấu `+` trở đi nữa.

3.10 Thủ tục `showClock`

Hàm này để hiển thị một đồng hồ đơn giản bao gồm thứ, ngày, tháng, năm và giờ, phút, giây lấy từ hệ thống (sử dụng các hàm `2Ah` và `2Ch` của ngắt `21h`).
Thứ tự thực hiện các chức năng của hàm này như sau:

- Hiện ra các thông báo cần thiết như `Hom nay la ...`, `Bay gio la ....`
- Ghi lại vị trí của của con trỏ mà tại đó cần hiển thị ngày và giờ. Vị trí để hiển thị ngày là ngay sau câu thông báo `Hom nay la`, còn vị trí hiển thị giờ là ngay sau câu thông báo `Bay gio la`.
- Thực hiện một vòng lặp vô hạn đến khi người dùng nhấn giữ nút `CTRL` thì thôi:
  - Di chuyển con trỏ đến vị trí cần hiển thị ngày. Đọc ngày của hệ thống và hiển thị nó ra màn hình. Ngày được in có định dạng tương tự như:
    `Thu 3, 05/05/2021`
  - Di chuyển con trỏ đến vị trí cần hiển thị giờ. Đọc giờ của hệ thống và hiển thị nó ra màn hình. Giờ được in có định dạng tương tự như:
    `15:23:01`.
  - Yêu cầu BIOS nằm chờ 1 giây. Để làm được như vậy, ta cần gán số 1 triệu (micro giây) vào thanh ghi tổ hợp `CX:AX` và gọi hàm số `86h` của ngắt `15h`.
    1 triệu ở hệ 16 là `000f 4240h`, vậy `CX = 0fh` và `AX = 4240h`.
  - Đọc dữ liệu của byte `40h:17h` từ `BIOS Data Area` tương tự như đầu chương trình để đọc trạng thái phím CTRL. Thực hiện shift right 3 lần để lấy bit biểu diễn trạng thái phím CTRL.

– Nếu bit đó là 1, thoát khỏi vòng lặp. Nếu không, tiếp tục lặp.

- Kết quả ta thu được có định dạng tương tự như sau:

```
Hom nay la Thu 3, 05/05/2021
Bay gio la 15:23:01
```

3.11 Biên dịch và chạy

Chương trình được dịch bằng trình hợp dịch `emu8086 v4.08` và chạy thử trong trình giả lập `DOSBox 0.74-3`.

Các chức năng đều hoạt động ổn định, không phát sinh bất cứ lỗi nào nghiêm trọng.

Đôi khi (có lẽ do trình giả lập) mà phím NumLock khi không nhấn vẫn bị xem là có nhấn hoặc ngược lại, phải thoát trình giả lập rồi mở lại mới khắc phục được.

Lưu ý: để tránh phát sinh lỗi hiển thị do con trỏ màn hình hay nhảy đi lung tung, thầy vui lòng luôn luôn chạy lệnh xoá màn hình `cls` trên shell của DOSBox trước khi chạy chương trình để tránh màn hình bị cuộn làm sai lệnh các toạ độ đã lưu.

Một số hình ảnh trong quá trình chạy thử *(trang sau)*:

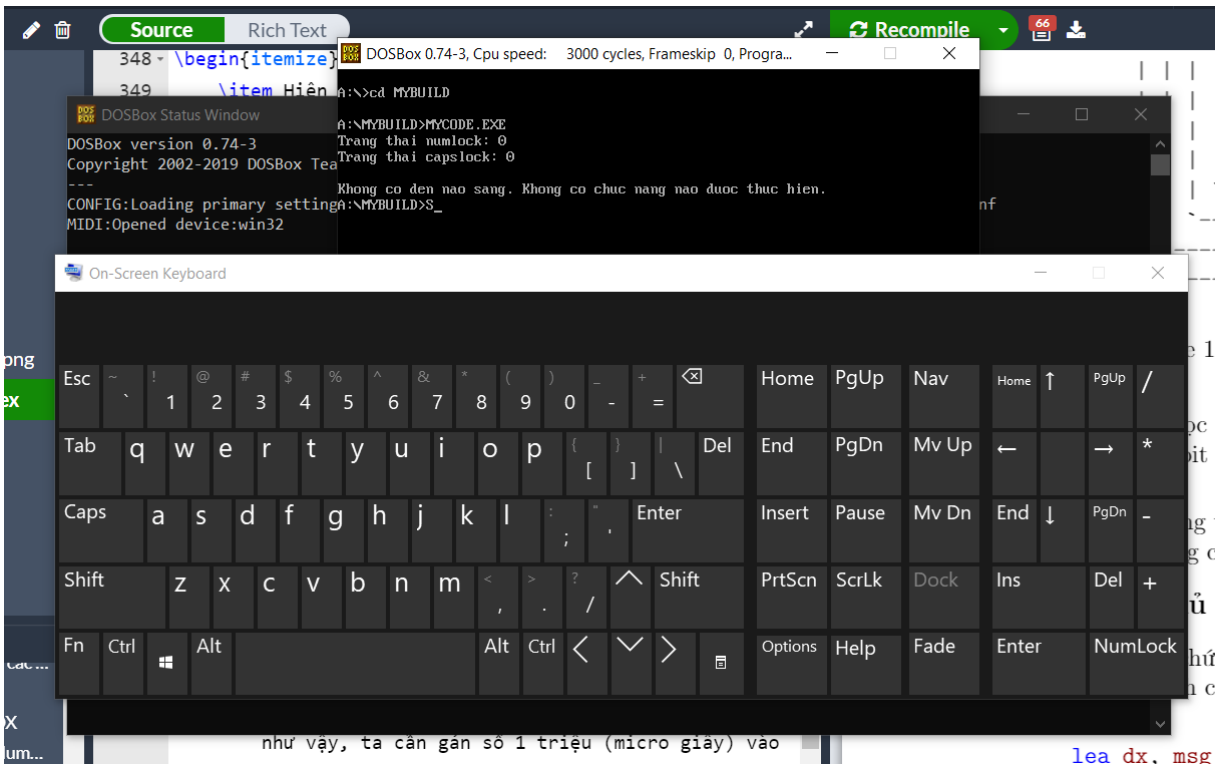

Figure 3: Khi không có phím nào được nhấn

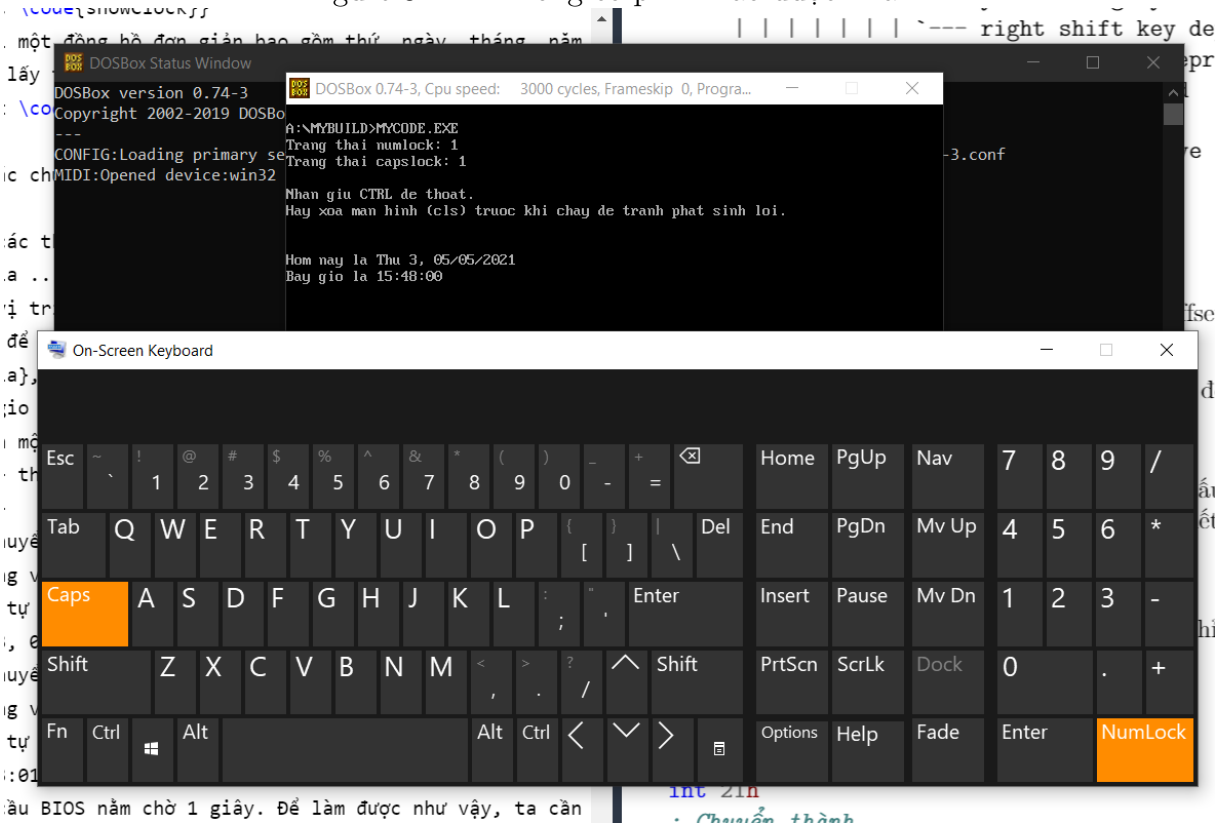

Figure 4: Khi cả 2 phím được nhấn

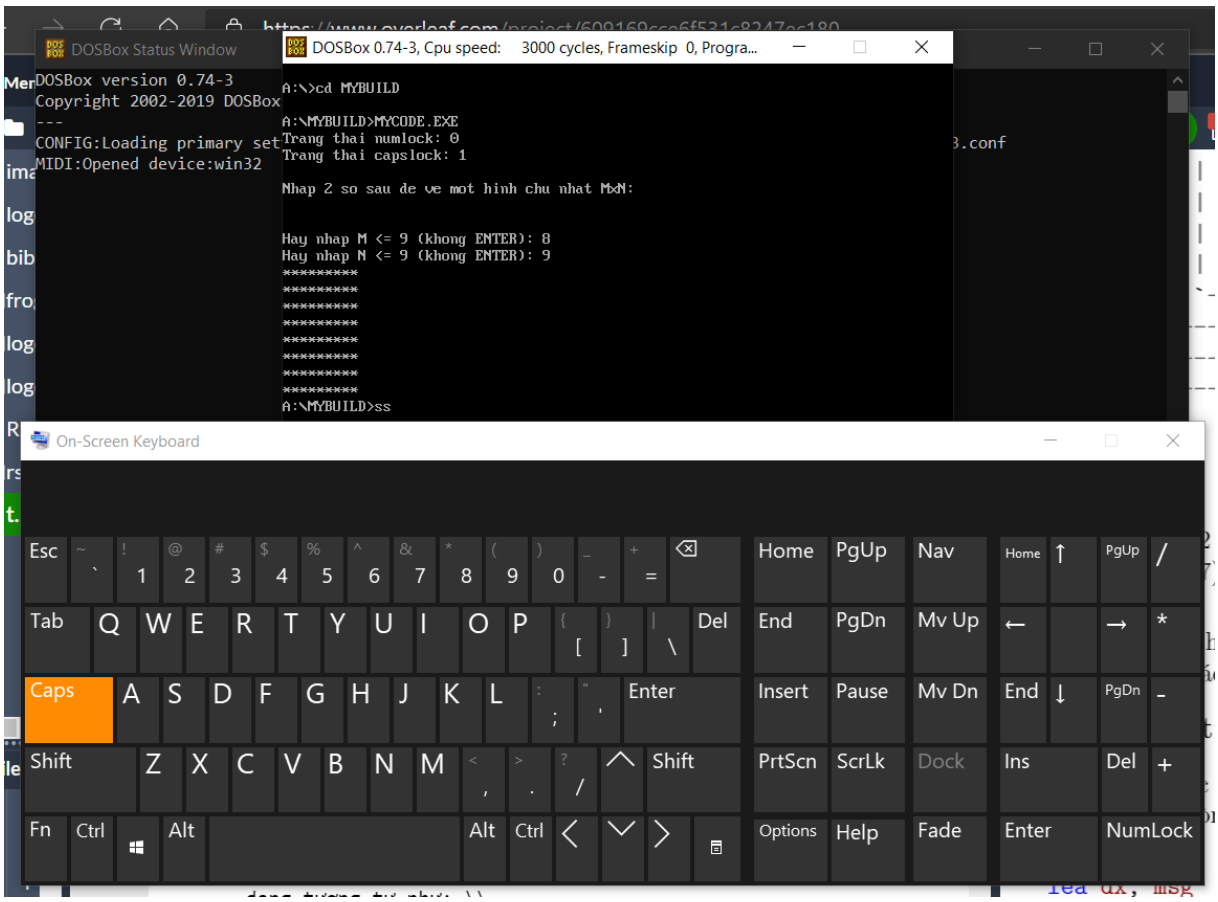

Figure 5: Khi chỉ có CapsLock được nhấn

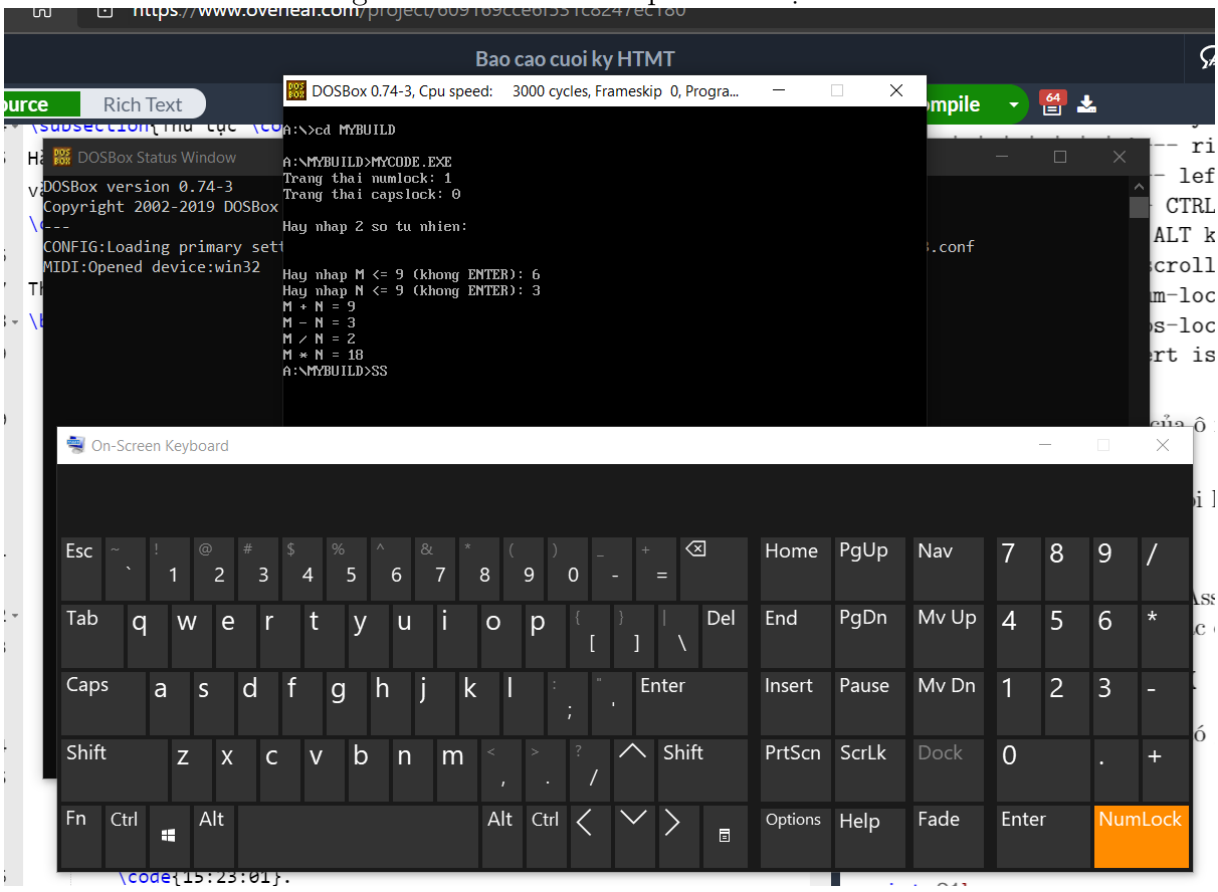

Figure 6: Khi chỉ có NumLock được nhấn

4 Tài liệu tham khảo

[1] J. Wyatt. 8086 instructions. (accessed: 04.05.2021). [Online]. Available: https://jbwyatt.com/253/emu/8086_instruction_set.html

[2] ——. Basic 8086 and dos interrupts that are currently supported by the emulator. (accessed: 04.05.2021). [Online]. Available: https://jbwyatt.com/253/emu/8086_bios_and_dos_interrupts.html

[3] T. H. Văn. Các slide bài giảng môn Hệ thống máy tính. [Online]. Available: Moodle

[4] StackOverflow. C Function to Convert float to byte array. (accessed: 12.05.2021), solution by Floris. [Online]. Available: https://stackoverflow.com/questions/24420246/c-function-to-convert-float-to-byte-array

[5] J. Wyatt. 8086 assembler tutorial for beginners (part 1). (accessed: 05.05.2021). [Online]. Available: https://jbwyatt.com/253/emu/asm_tutorial_01.html

[6] ——. 8086 assembler tutorial for beginners (part 9). (accessed: 05.05.2021). [Online]. Available: https://jbwyatt.com/253/emu/asm_tutorial_09.html

[7] D. Jurgens. BDA - BIOS Data Area - PC Memory Map. (accessed: 04.05.2021). [Online]. Available: https://stanislavs.org/helppc/bios_data_area.html

[8] Wikipedia. Eucledian division. (accessed: 05.05.2021). [Online]. Available: https://en.wikipedia.org/wiki/Euclidean_division

[9] Trần Đan Thư et al., *Kỹ thuật lập trình*. NXB Khoa học và Kỹ thuật, 2019.

[10] ——, *Lập trình hướng đối tượng*. NXB Khoa học và Kỹ thuật, 2019.

Hết