HSGS Sec Day 03

C Pointers and Memory

Agenda

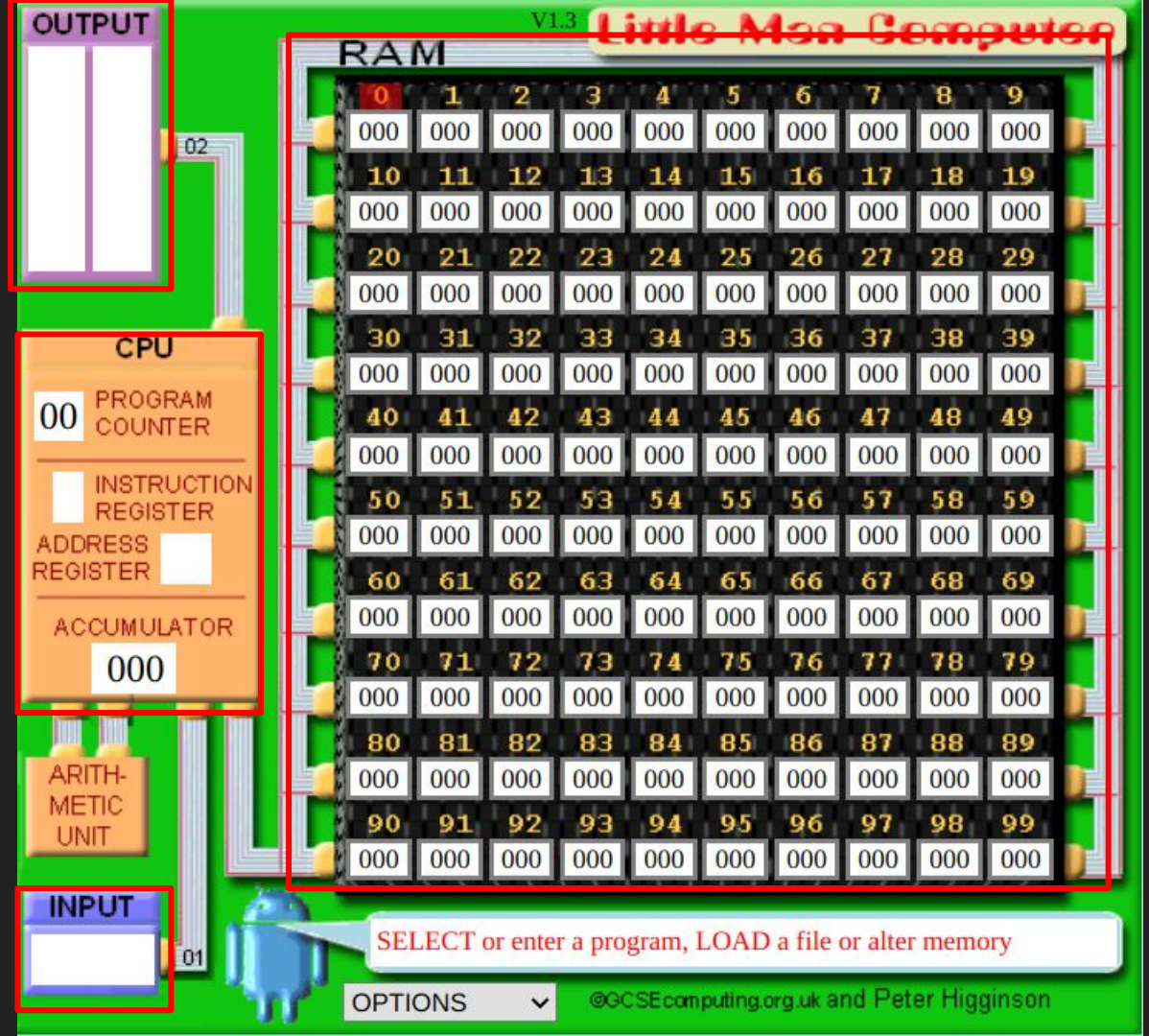
1. Quiz
2. Chữa bài tập
3. C Pointers
4. Memory
    a. Endianness, representation
    b. Asm operations

Agenda

1. Quiz
2. Chữa bài tập
3. C Pointers
4. Memory
    a. Endianness, representation
    b. Asm operations

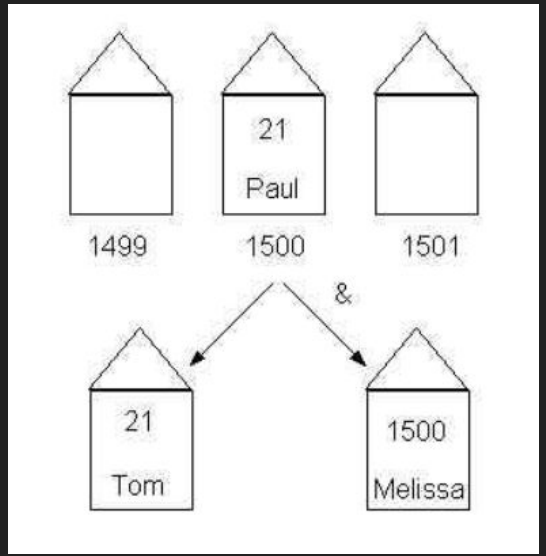
Agenda

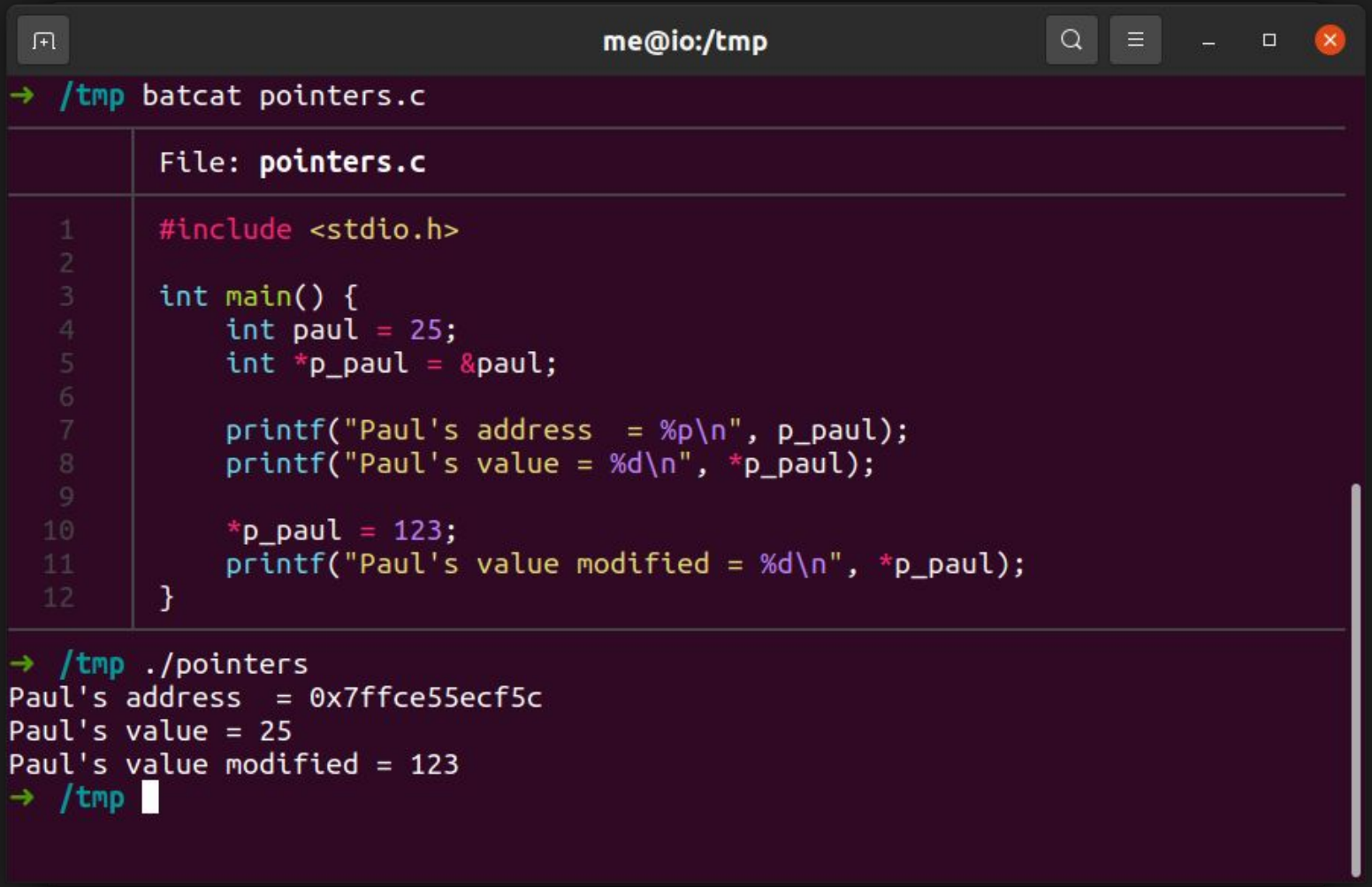
1. Quiz
2. Chữa bài tập
3. C Pointers
4. Memory
    a. Endianness, representation
    b. Asm operations


OUTPUT
02
V1.3
Little Man Computer
RAM
0 1 2 3 4 5 6 7 8 9
000 000 000 000 000 000 000 000 000 000
10 11 12 13 14 15 16 17 18 19
000 000 000 000 000 000 000 000 000 000
20 21 22 23 24 25 26 27 28 29
000 000 000 000 000 000 000 000 000 000
30 31 32 33 34 35 36 37 38 39
000 000 000 000 000 000 000 000 000 000
40 41 42 43 44 45 46 47 48 49
000 000 000 000 000 000 000 000 000 000
50 51 52 53 54 55 56 57 58 59
000 000 000 000 000 000 000 000 000 000
60 61 62 63 64 65 66 67 68 69
000 000 000 000 000 000 000 000 000 000
70 71 72 73 74 75 76 77 78 79
000 000 000 000 000 000 000 000 000 000
80 81 82 83 84 85 86 87 88 89
000 000 000 000 000 000 000 000 000 000
90 91 92 93 94 95 96 97 98 99
000 000 000 000 000 000 000 000 000 000
CPU
00
PROGRAM COUNTER
INSTRUCTION REGISTER
ADDRESS REGISTER
ACCUMULATOR
000
ARITH-METIC UNIT
INPUT
01
SELECT or enter a program, LOAD a file or alter memory
OPTIONS
©GCSEcomputing.org.uk and Peter Higginson

C Pointers

- Pointers là một kiểu dữ liệu chứa địa chỉ của các dữ liệu khác
- Ví dụ:
  - Paul sống ở ngôi nhà số 1500
  - Ngôi nhà này chứa số 21 (int paul = 21)
  - ⇒ Địa chỉ của Paul (21) là 1500 (1500 == &paul)
  - Có một ngôi nhà khác chứa địa chỉ nhà của Paul (int *melissa = &paul)
  - ⇒ Ngôi nhà này là pointer tới nhà Paul (int *)




me@io:/tmp
→ /tmp batcat pointers.c
File: pointers.c
1 #include <stdio.h>
2
3 int main() {
4 int paul = 25;
5 int *p_paul = &paul;
6
7 printf("Paul's address = %p\n", p_paul);
8 printf("Paul's value = %d\n", *p_paul);
9
10 *p_paul = 123;
11 printf("Paul's value modified = %d\n", *p_paul);
12 }
→ /tmp ./pointers
Paul's address = 0x7ffce55ecf5c
Paul's value = 25
Paul's value modified = 123
→ /tmp

C Pointers

- Vì pointer cũng là 1 kiểu dữ liệu nên có thể tồn tại pointer tới pointer

C Pointers Arithmetic

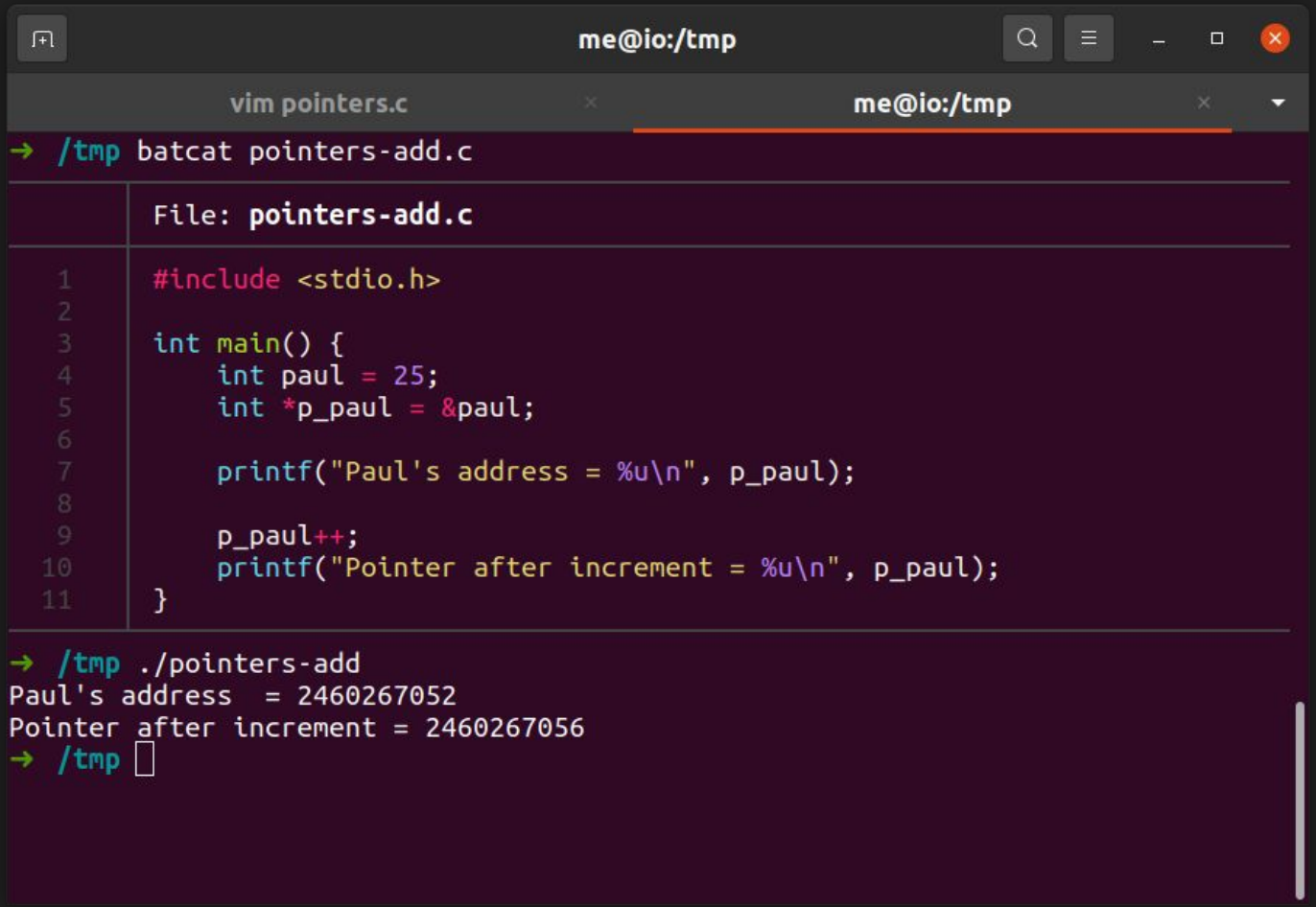
- Các phép toán với pointers khác phép toán thông thường
- Phép tăng/giảm pointers sẽ khiến pointer trỏ tới phần tử trước/sau của cùng một loại dữ liệu

  Giả sử int *p = 1000;

  Kiểu int có size = 4;

  p++ ⇒ p == ???

  p == (1000 + 4) == 1004


me@io:/tmp
vim pointers.c
me@io:/tmp
→ /tmp batcat pointers-add.c
File: pointers-add.c
1 #include <stdio.h>
2
3 int main() {
4 int paul = 25;
5 int *p_paul = &paul;
6
7 printf("Paul's address = %u\n", p_paul);
8
9 p_paul++;
10 printf("Pointer after increment = %u\n", p_paul);
11 }
→ /tmp ./pointers-add
Paul's address = 2460267052
Pointer after increment = 2460267056
→ /tmp

C Pointers and Array

- Pointers và mảng không hoàn toàn giống nhau
- Một số điểm chung
  - int a[5]; int *p = a
  - Biến a và pointer p trỏ cùng một địa chỉ bộ nhớ a[0]
  - p++ ⇒ Trỏ tới phần tử a[1];
- Sự khác biệt
  - Không thể gán kiểu pointer cho 1 biến array
  - int a[5]; int *p
  - Không thể gán a=p;


me@io:/tmp
me@io:/tmp
me@io:/tmp
→ /tmp batcat pointer-array.c
File: pointer-array.c
1 #include <stdio.h>
2
3 int main() {
4 int a[5] = {11, 22, 33, 44, 55};
5 int *p = a;
6
7 printf("Value of p = %d\n", *p);
8 p++;
9 printf("Now value of p = %d\n", *p);
10 }
→ /tmp ./pointer-array
Value of p = 11
Now value of p = 22
→ /tmp

C Pointers

- Khai báo kiểu pointer: int *, char *
  - Lưu ý: int *a, b sẽ khai báo pointer a và int b
  - int *a, *b sẽ khai báo 2 pointer a và b
- Thao tác lấy địa chỉ: int x = 25; int *p_x = &x;
- Thao tác chỉnh sửa nội dung/giá trị bộ nhớ: int *p_x = &x; *p_x = 100;
  - Còn gọi là dereference
- Toán với pointer: int *p_x = 100; p_x ++; p_x += 100
- In ra kiểu pointer: printf("%p");

Memory Allocation

- Khi không biết trước lượng bộ nhớ cần cấp phát, cần thực hiện cấp phát bộ nhớ động (malloc)
- Sau khi sử dụng bộ nhớ đã cấp phát, cần phải trả lại cho hệ thống (free)
- Ví dụ

```
int *x = (int *) malloc(sizeof(int)); // Cấp phát bộ nhớ cho 1 biến int

int *y = (int *) malloc(100 * sizeof(int)); // Cấp phát bộ nhớ cho 100 biến int

free(x); // Sau khi dùng, phải trả lại bộ nhớ cho hệ thống
```

- malloc trả về kiểu (void *), cần phải ép kiểu (cast) sang kiểu pointer cần dùng

Agenda

1. Quiz
2. Chữa bài tập
3. C Pointers
4. Memory
    a. Endianness, representation
    b. Asm operations

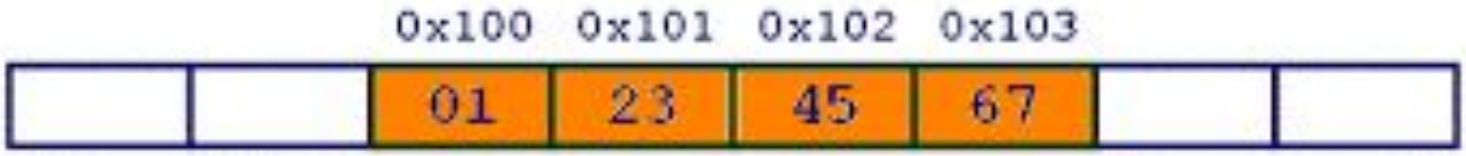
Endianness

- Giả sử ta có số nguyên 01234567 cần lưu trong bộ nhớ địa chỉ 100. Mỗi địa chỉ lưu 1 chữ số. Vậy thì tại địa chỉ 100 có chữ số nào?

  0??

  7??

⇒ Endianness (tính kết thúc) sẽ quyết định điều này



Big Endian

Little Endian

Cách truy cập bộ nhớ trong ASM

Các toán tử bộ nhớ có dạng như sau

- [number] ⇒ [1234]
  - Trỏ tới ô nhớ số 1234
  - 1234 ⇒ displacement
- [reg] ⇒ [rbx]
  - Nếu rbx == 16, trỏ tới ô nhớ số 16
  - rbx ⇒ base address
- [reg1 + reg2*scale] ⇒ [rbx + rdi*2]
  - reg1 ⇒ base address
  - reg2 ⇒ index
  - scale ⇒ size của các kiểu cơ bản, scale = 1, 2, 4 hoặc 8

Cách truy cập bộ nhớ trong ASM

Các toán tử bộ nhớ có dạng như sau

- [reg + number] ⇒ [rbx + 1]
- [reg1 + reg2*scale + number] ⇒ [rbx + rdi * 1 + 16]

Cách truy cập bộ nhớ trong ASM

- Nhớ lại các instructions đã học, những instruction nào có operand là mem thì có thể sử dụng những cách truy cập đã nêu
- Ví dụ
  - mov dword [rbx], 1234 ==> Nếu rbx == 16, lưu số 1234 vào ô nhớ tại vị trí 16
- Instruction mới: LEA
  - LEA thực hiện việc tính toán nhanh trên register để phục vụ truy cập bộ nhớ
  - lea rax, [rdi + 8] ==> rax = rdi + 8
  - Lưu ý: mov rax, rdi+8 => Không hợp lệ
- Giả sử rdi là pointer tới 1 mảng long long (8 byte/phần tử)
  - lea rax, [rdi + 8] trỏ tới phần tử tiếp theo sau rdi

Resources

[1] Pointers Introduction

[2] C - Pointers

[3] NASM Tutorial

[4] Endianness Explained With an Egg - Computerphile

[5] LEA — Load Effective Address