Báo cáo bài tập tuần 3

Môi trường thực hiện: windows, linux 64 bit

Linux: sử dụng nasm để build

- Nasm –f elf 64 file.asm: biên dịch để tạo file object
- Ld file.o –o file: static link để link file object với file thực thi

Windows: sử dụng masm64 để build

- C:\masm32\bin64\ml64 /c file.asm: /c để không link với file thực thi
- C:\masm32\bin64\link /subsystem:console /entry:main: tạo chương trình chạy trên console

Bài 9: tìm giá trị lớn nhất, nhỏ nhất trong mảng

linux:

1. **Section .data**
   - 1.1. Chuỗi "input n" để in ra màn hình
   - 1.2. Chuỗi "array[" để in ra màn hình
   - 1.3. Chuỗi "] = " để in ra màn hình
   - 1.4. Chuỗi "max = " để in ra màn hình
   - 1.5. Chuỗi "min = " để in ra màn hình
   - 1.6. Chuỗi "invalid!!" để in ra màn hình
   - 1.7. Chuỗi "overflow!!" để in ra màn hình
   - 1.8. Endline để in ký tự kết thúc
2. **Section .bss**
   - 2.1. **Arr** để lưu trữ mảng
   - 2.2. **n** để lưu trữ số lượng của mảng
   - 2.3. **min** để lưu trữ giá trị nhỏ nhất trong mảng
   - 2.4. **max** để lưu trữ giá trị lớn nhất trong mảng
   - 2.5. **inp** để lưu trữ 1 phần tử của mảng
3. **Section .text bao gồm code thực thi là entry point và lable _start**
   - **3.1. Các hàm nhập xuất sử dụng syscall**
     - **3.1.1. PrintString**
       - Rax = 1: lưu trữ opcode sys_write

- Rdi = 1: là stdout file của descriptor
- Rsi là địa chỉ chứa chuỗi in ra màn hình
- Rdx là số lượng ký tự của chuỗi

3.1.2. ReadString

- Rax = 0: lưu trữ opcode sys_write
- Rdi = 0: là stdin file của descriptor
- Rsi là địa chỉ chứa chuỗi đọc vào
- Rdx là số lượng ký tự tối đa của chuỗi nhập vào

3.2. Luồng thực thi chương trình

3.2.1. Các hàm chức năng được sử dụng

- Hàm **checkNum** kiểm tra xem chuỗi tham số có phải là số không bằng cách kiểm tra từng ký tự trong khoảng từ 0x30-0x39.
- Hàm **atoi** chuyển chuỗi tham số thành số tương ứng(sau khi đã checkNum) bằng cách tách từng số VD: 1234 = ((1 * 10 + 2) * 10 + 3)* 10 + 4. Đồng thời sau khi chuyển cũng kiểm tra tràn 31 bit bằng cách dịch phải 24 bit để kiểm tra xem 2 byte đầu tiên có vượt quá 127 hay không kết quả số sau khi đổi trả về rax.
- Hàm **getlen** trả về độ dài của chuỗi tham số bằng cách xét từng ký tự cho đến 0 thì cộng thêm 1 vào rax.
- Hàm **findMax, findMin** thực hiện tìm giá trị lớn nhất, nhỏ nhất trong mảng bằng cách so sánh từ giá trị đầu tiên với ký tự ngay sau nó cho đến hết mảng.

3.2.2. Luồng thực hiện chương trình

- Đầu tiên nhập giá trị của n.
- Tiếp theo nhập từng phần tử cho mảng và đổi thành số với hàm **atoi**, trong đó có kiểm tra xem giá trị nhập vào có vượt quá giá trị số dương hay không bằng cách dịch phải 28 bit và kiểm tra xem byte đầu tiên có vượt quá 7 không.
- Cuối cùng sau khi đã có dữ liệu cho mảng, tiến hành tìm giá trị lớn nhất, nhỏ nhất trong mảng với hàm **findMax**, **findMin** rồi in ra màn hình.

Windows

1. Section .data

1.1. Chuỗi "input n" để in ra màn hình
1.2. Chuỗi "array[" để in ra màn hình
1.3. Chuỗi "] = " để in ra màn hình
1.4. Chuỗi "max = " để in ra màn hình
1.5. Chuỗi "min = " để in ra màn hình
1.6. Chuỗi "invalid!!" để in ra màn hình
1.7. Chuỗi "overflow!!" để in ra màn hình
1.8. Endline để in ký tự kết thúc

2. Section .data?

2.1. **Arr** để lưu trữ mảng
2.2. **n** để lưu trữ số lượng của mảng
2.3. **min** để lưu trữ giá trị nhỏ nhất trong mảng
2.4. **max** để lưu trữ giá trị lớn nhất trong mảng
2.5. **inp** để lưu trữ 1 phần tử của mảng.

3. Section .code bao gồm code thực thi với entry point là main proc

3.1. Các hàm nhập xuất sử dụng windows api.

3.1.1. WriteString

- Rcx: nStdHandle
- Rdx: offset chuỗi cần ghi
- R8: số lượng ký tự in (lấy theo độ dài chuỗi trả về từ lstrlen)
- R9: offset cho số lương ký tự in- lấy từ biến của hàm WriteString(rbp - 8)

3.1.2. ReadString

- Rcx: nStdHandle
- Rdx: offset chuỗi cần đọc
- R8: số lượng ký tự đọc(gán cứng là maxchar = 256)
- R9: offset cho số lương ký tự in- lấy từ biến của hàm WriteString(rbp - 8)

3.2. Luồng thực thi của chương trình

3.2.1. Các hàm chức năng được sử dụng

- Hàm **checkNum** kiểm tra xem chuỗi tham số có phải là số không bằng cách kiểm tra từng ký tự trong khoảng từ 0x30-0x39.

- **Ád** Hàm **atoi** chuyển chuỗi tham số thành số tương ứng(sau khi đã checkNum) bằng cách tách từng số VD: 1234 = ((1 * 10 + 2) * 10 + 3)* 10 + 4. Đồng thời sau khi chuyển cũng kiểm tra tràn 31 bit bằng cách dịch phải 24 bit để kiểm tra xem 2 byte đầu tiên có vượt quá 127 hay không kết quả số sau khi đổi trả về rax.
- Hàm **getlen** trả về độ dài của chuỗi tham số bằng cách xét từng ký tự cho đến 0 thì cộng thêm 1 vào rax.
- Hàm **findMax, findMin** thực hiện tìm giá trị lớn nhất, nhỏ nhất trong mảng bằng cách so sánh từ giá trị đầu tiên với ký tự ngay sau nó cho đến hết mảng.

3.2.2. Luồng thực hiện của chương trình

- Đầu tiên nhập giá trị của n.
- Tiếp theo nhập từng phần tử cho mảng và đổi thành số với hàm **atoi**, trong đó có kiểm tra xem giá trị nhập vào có vượt quá giá trị số dương hay không bằng cách dịch phải 28 bit và kiểm tra xem byte đầu tiên có vượt quá 7 không.
- Cuối cùng sau khi đã có dữ liệu cho mảng, tiến hành tìm giá trị lớn nhất, nhỏ nhất trong mảng với hàm **findMax**, **findMin** rồi in ra màn hình.

Bài 10: tính tổng phần tử chẵn, lẻ trong mảng

linux:

1. Section .data

1.1. Chuỗi "input n" để in ra màn hình
1.2. Chuỗi "array[" để in ra màn hình
1.3. Chuỗi "] = " để in ra màn hình
1.4. Chuỗi "sum of odd numbers = " để in ra màn hình
1.5. Chuỗi "sum of even numbers = " để in ra màn hình
1.6. Chuỗi "invalid!!" để in ra màn hình
1.7. Chuỗi "overflow!!" để in ra màn hình
1.8. Endline để in ký tự kết thúc

2. Section .bss

2.1. **Arr** để lưu trữ mảng

2.2. **n** để lưu trữ số lượng của mảng

2.3. **inp** để lưu trữ 1 phần tử của mảng

3. Section .text bao gồm code thực thi là entry point và lable _start

3.1. Các hàm nhập xuất sử dụng syscall

3.1.1. PrintString

- Rax = 1: lưu trữ opcode sys_write
- Rdi = 1: là stdout file của descriptor
- Rsi là địa chỉ chứa chuỗi in ra màn hình
- Rdx là số lượng ký tự của chuỗi

3.1.2. ReadString

- Rax = 0: lưu trữ opcode sys_write
- Rdi = 0: là stdin file của descriptor
- Rsi là địa chỉ chứa chuỗi đọc vào
- Rdx là số lượng ký tự tối đa của chuỗi nhập vào

3.2. Luồng thực thi chương trình

3.2.1. Các hàm chức năng được sử dụng

- Hàm **checkNum** kiểm tra xem chuỗi tham số có phải là số không bằng cách kiểm tra từng ký tự trong khoảng từ 0x30-0x39.
- Hàm **atoi** chuyển chuỗi tham số thành số tương ứng(sau khi đã checkNum) bằng cách tách từng số VD: 1234 = ((1 * 10 + 2) * 10 + 3)* 10 + 4. Đồng thời sau khi chuyển cũng kiểm tra tràn 31 bit bằng cách dịch phải 24 bit để kiểm tra xem 2 byte đầu tiên có vượt quá 127 hay không kết quả số sau khi đổi trả về rax.
- Hàm **getlen** trả về độ dài của chuỗi tham số bằng cách xét từng ký tự cho đến 0 thì cộng thêm 1 vào rax.
- Hàm **sumOdd, sumEven** thực hiện tìm giá trị lẻ, chẵn trong mảng bằng cách thực hiện toán tử and từng phần tử với 1, nếu phần tử đang xet bằng 1 thì là số lẻ, ngược lại là số chẵn, sau mỗi lần tìm phần tử chẵn, lẻ trong mảng thì cộng với **r14** cho đến hết mảng và in giá trị trong **r14** ra màn hình.

3.2.2. Luồng thực hiện chương trình

- Đầu tiên nhập giá trị của n.

- Tiếp theo nhập từng phần tử cho mảng và đổi thành số với hàm **atoi**, trong đó có kiểm tra xem giá trị nhập vào có vượt quá giá trị số dương hay không bằng cách dịch phải 28 bit và kiểm tra xem byte đầu tiên có vượt quá 7 không.
- Cuối cùng sau khi đã có dữ liệu cho mảng, tiến hành tìm tổng các giá trị chẵn, lẻ trong mảng và in ra màn hình.

Windows

1. Section .data

1.1. Chuỗi "input n" để in ra màn hình
1.2. Chuỗi "array[" để in ra màn hình
1.3. Chuỗi "] = " để in ra màn hình
1.4. Chuỗi "max = " để in ra màn hình
1.5. Chuỗi "min = " để in ra màn hình
1.6. Chuỗi "invalid!!" để in ra màn hình
1.7. Chuỗi "overflow!!" để in ra màn hình
1.8. Endline để in ký tự kết thúc

2. Section .data?

2.1. **Arr** để lưu trữ mảng
2.2. **n** để lưu trữ số lượng của mảng
2.3. **min** để lưu trữ giá trị nhỏ nhất trong mảng
2.4. **max** để lưu trữ giá trị lớn nhất trong mảng
2.5. **inp** để lưu trữ 1 phần tử của mảng.

3. Section .code bao gồm code thực thi với entry point là main proc

3.1. Các hàm nhập xuất sử dụng windows api.

3.1.1. WriteString

- Rcx: nStdHandle
- Rdx: offset chuỗi cần ghi
- R8: số lượng ký tự in (lấy theo độ dài chuỗi trả về từ lstrlen)
- R9: offset cho số lương ký tự in- lấy từ biến của hàm WriteString(rbp - 8)

3.1.2. ReadString

- Rcx: nStdHandle
- Rdx: offset chuỗi cần đọc

- R8: số lượng ký tự đọc(gán cứng là maxchar = 256)
- R9: offset cho số lương ký tự in- lấy từ biến của hàm WriteString(rbp - 8)

3.2. Luồng thực thi của chương trình

3.2.1. Các hàm chức năng được sử dụng

- Hàm **checkNum** kiểm tra xem chuỗi tham số có phải là số không bằng cách kiểm tra từng ký tự trong khoảng từ 0x30-0x39.
- Hàm **atoi** chuyển chuỗi tham số thành số tương ứng(sau khi đã checkNum) bằng cách tách từng số VD: 1234 = ((1 * 10 + 2) * 10 + 3)* 10 + 4. Đồng thời sau khi chuyển cũng kiểm tra tràn 31 bit bằng cách dịch phải 24 bit để kiểm tra xem 2 byte đầu tiên có vượt quá 127 hay không kết quả số sau khi đổi trả về rax.
- Hàm **getlen** trả về độ dài của chuỗi tham số bằng cách xét từng ký tự cho đến 0 thì cộng thêm 1 vào rax.
- Hàm **findMax, findMin** thực hiện tìm giá trị lớn nhất, nhỏ nhất trong mảng bằng cách so sánh từ giá trị đầu tiên với ký tự ngay sau nó cho đến hết mảng.

3.2.2. Luồng thực hiện của chương trình

- Đầu tiên nhập giá trị của n.
- Tiếp theo nhập từng phần tử cho mảng và đổi thành số với hàm **atoi**, trong đó có kiểm tra xem giá trị nhập vào có vượt quá giá trị số dương hay không bằng cách dịch phải 28 bit và kiểm tra xem byte đầu tiên có vượt quá 7 không.
- Cuối cùng sau khi đã có dữ liệu cho mảng, tiến hành tìm giá trị lớn nhất, nhỏ nhất trong mảng với hàm **findMax**, **findMin** rồi in ra màn hình.

Bài 11: chương trình mô phỏng máy tính đơn giản

linux:

1. Section .data

1.1. Chuỗi "chon phep tinh" để in ra màn hình

1.2. Chuỗi "1. Phep cong" để in ra màn hình

1.3. Chuỗi "2. Phep tru" để in ra màn hình
1.4. Chuỗi "3. Phep nhan" để in ra màn hình
1.5. Chuỗi "4. Phep chia" để in ra màn hình
1.6. Chuỗi "nhap so 1 = " để in ra màn hình
1.7. Chuỗi "nhap so 2 = " để in ra màn hình
1.8. Chuỗi "invalid!!" để in ra màn hình
1.9. Chuỗi "overflow!!" để in ra màn hình
1.10. Ký tự "-" để in ra màn hình
1.11. Endline để in ký tự kết thúc

2. Section .bss

2.1. **select** để lưu trữ giá trị chọn
2.2. **num1** để lưu trữ số thứ nhất
2.3. **num2** để lưu trữ số thứ hai
2.4. **res** để lưu trữ kết quả của phép tính

3. Section .text bao gồm code thực thi là entry point và lable _start

3.1. Các hàm nhập xuất sử dụng syscall

3.1.1. PrintString

- Rax = 1: lưu trữ opcode sys_write
- Rdi = 1: là stdout file của descriptor
- Rsi là địa chỉ chứa chuỗi in ra màn hình
- Rdx là số lượng ký tự của chuỗi

3.1.2. ReadString

- Rax = 0: lưu trữ opcode sys_write
- Rdi = 0: là stdin file của descriptor
- Rsi là địa chỉ chứa chuỗi đọc vào
- Rdx là số lượng ký tự tối đa của chuỗi nhập vào

3.2. Luồng thực thi chương trình

3.2.1. Các hàm chức năng được sử dụng

- Hàm **PrintMenu** thực hiện in menu lựa chọn phép tính cộng, trừ, nhân, chia ra màn hình.
- Hàm **checkSelect** kiểm tra xem giá trị nhập vào có phải là một trong 4 giá trị 1, 2, 3, 4 hay không. Nếu không hợp lệ thì thoát chương trình.

- Hàm **checkNum** kiểm tra xem chuỗi tham số có phải là số không bằng cách kiểm tra từng ký tự trong khoảng từ 0x30-0x39. Ngoài ra còn kiểm tra xem chuỗi có ký tự '-' không, nếu có thì chuỗi là số âm.
- Hàm **atoi** chuyển chuỗi tham số thành số tương ứng(sau khi đã checkNum) bằng cách tách từng số VD: 1234 = ((1 * 10 + 2) * 10 + 3)* 10 + 4 ,nếu trong chuỗi có ký tự '-' thì chuyển thành số âm. Đồng thời sau khi chuyển cũng kiểm tra tràn 32 bit bằng cách dịch phải 32 bit để kiểm tra xem 2 byte đầu tiên có vượt quá khác 0 hay không kết quả số sau khi đổi trả về rax.
- Hàm **getlen** trả về độ dài của chuỗi tham số bằng cách xét từng ký tự cho đến 0 thì cộng thêm 1 vào rax.
- Hàm **calc** nhận lựa chọn từ hàm checkSelect và thực hiện phép tính tương ứng.
    - Phép cộng thực hiện cộng 2 số, nếu thấy 2 byte đầu tiên khác 0x7f thì là số âm => thêm ký tự '-'.
    - Phép trừ thực hiện từ 2 số, nếu thấy 2 byte đầu tiên khác 0x7f thì là số âm => thêm ký tự '-'.
    - Phép nhân thực hiện nhân 2 số và có mở rộng thanh ghi với cqo tránh tràn số.
    - Phép chia nhân 2 số và có mở rộng thanh ghi tránh trường hợp tràn số.

3.2.2. Luồng thực hiện chương trình

- Đầu in giá menu của chương trình cho phép chọn 1, 2, 3, 4 làm 4 phép tính tương ứng.
- Tiếp theo cho phép nhập 2 số để thực hiện tính toán
- Cuối cùng tính toán 2 số theo phép tính lựa chọn với 2 số đã nhập vào

Windows

1. Section .data

1.1. Chuỗi "chon phep tinh" để in ra màn hình

1.2. Chuỗi "1. Phep cong" để in ra màn hình

1.3. Chuỗi "2. Phep tru" để in ra màn hình
1.4. Chuỗi "3. Phep nhan" để in ra màn hình
1.5. Chuỗi "4. Phep chia" để in ra màn hình
1.6. Chuỗi "nhap so 1 = " để in ra màn hình
1.7. Chuỗi "nhap so 2 = " để in ra màn hình
1.8. Chuỗi "invalid!!" để in ra màn hình
1.9. Chuỗi "overflow!!" để in ra màn hình
1.10. Ký tự "-" để in ra màn hình
1.11. Endline để in ký tự kết thúc

2. Section .data?

2.1. **select** để lưu trữ giá trị chọn
2.2. **num1** để lưu trữ số thứ nhất
2.3. **num2** để lưu trữ số thứ hai
2.4. **res** để lưu trữ kết quả của phép tính

3. Section .code bao gồm code thực thi với entry point là main proc

3.1. Các hàm nhập xuất sử dụng windows api.

3.1.1. WriteString

- Rcx: nStdHandle
- Rdx: offset chuỗi cần ghi
- R8: số lượng ký tự in (lấy theo độ dài chuỗi trả về từ lstrlen)
- R9: offset cho số lương ký tự in- lấy từ biến của hàm WriteString(rbp - 8)

3.1.2. ReadString

- Rcx: nStdHandle
- Rdx: offset chuỗi cần đọc
- R8: số lượng ký tự đọc(gán cứng là maxchar = 256)
- R9: offset cho số lương ký tự in- lấy từ biến của hàm WriteString(rbp - 8)

3.2. Luồng thực thi của chương trình

3.2.1. Các hàm chức năng được sử dụng

- Hàm **PrintMenu** thực hiện in menu lựa chọn phép tính cộng, trừ, nhân, chia ra màn hình.

- Hàm **checkSelect** kiểm tra xem giá trị nhập vào có phải là một trong 4 giá trị 1, 2, 3, 4 hay không. Nếu không hợp lệ thì thoát chương trình.

- Hàm **checkNum** kiểm tra xem chuỗi tham số có phải là số không bằng cách kiểm tra từng ký tự trong khoảng từ 0x30-0x39. Ngoài ra còn kiểm tra xem chuỗi có ký tự '-' không, nếu có thì chuỗi là số âm.
- Hàm **atoi** chuyển chuỗi tham số thành số tương ứng(sau khi đã checkNum) bằng cách tách từng số VD: 1234 = ((1 * 10 + 2) * 10 + 3)* 10 + 4 ,nếu trong chuỗi có ký tự '-' thì chuyển thành số âm. Đồng thời sau khi chuyển cũng kiểm tra tràn 32 bit bằng cách dịch phải 32 bit để kiểm tra xem 2 byte đầu tiên có vượt quá khác 0 hay không kết quả số sau khi đổi trả về rax.
- Hàm **getlen** trả về độ dài của chuỗi tham số bằng cách xét từng ký tự cho đến 0 thì cộng thêm 1 vào rax.
- Hàm **calc** nhận lựa chọn từ hàm checkSelect và thực hiện phép tính tương ứng.
    - Phép cộng thực hiện cộng 2 số, nếu thấy 2 byte đầu tiên khác 0x7f thì là số âm => thêm ký tự '-'.
    - Phép trừ thực hiện từ 2 số, nếu thấy 2 byte đầu tiên khác 0x7f thì là số âm => thêm ký tự '-'.
    - Phép nhân thực hiện nhân 2 số và có mở rộng thanh ghi với cqo tránh tràn số.
    - Phép chia nhân 2 số và có mở rộng thanh ghi tránh trường hợp tràn số.

3.2.2. Luồng thực hiện của chương trình

- Đầu in giá menu của chương trình cho phép chọn 1, 2, 3, 4 làm 4 phép tính tương ứng.
- Tiếp theo cho phép nhập 2 số để thực hiện tính toán
- Cuối cùng tính toán 2 số theo phép tính lựa chọn với 2 số đã nhập vào