Introduction, arithmetics, registers

Agenda

1. Introduction
    a. Giới thiệu về HSGS Sec
    b. Nội dung khóa học
    c. Nội quy học tập
2. Giới thiệu về assembly language (asm)
3. Các phép tính toán cơ bản với asm
    a. Registers
    b. add, sub
    c. mul, div, mod

HSGS Sec Là Gì?

- HSGS Sec xuất phát là một chuỗi các buổi học dạy về một số chủ đề trong ngành an toàn thông tin (cybersecurity)
- Lần đầu tổ chức vào năm 2016
- HSGS Sec 2021 được tổ chức theo hình thức mới hoàn toàn

Nội Dung Khóa Học

- Lập trình cơ bản với assembly language x86-64
- Cơ bản về cơ chế hoạt động của CPU máy tính và bộ nhớ máy tính
- Cách hoạt động của một chương trình máy tính
- Giới thiệu về dịch ngược phần mềm và lỗi lập trình

Nội Quy Học Tập

- Không nói chuyện riêng trong khi học, đặc biệt với lớp học offline
- Vậy thôi đủ rồi :D

Tips

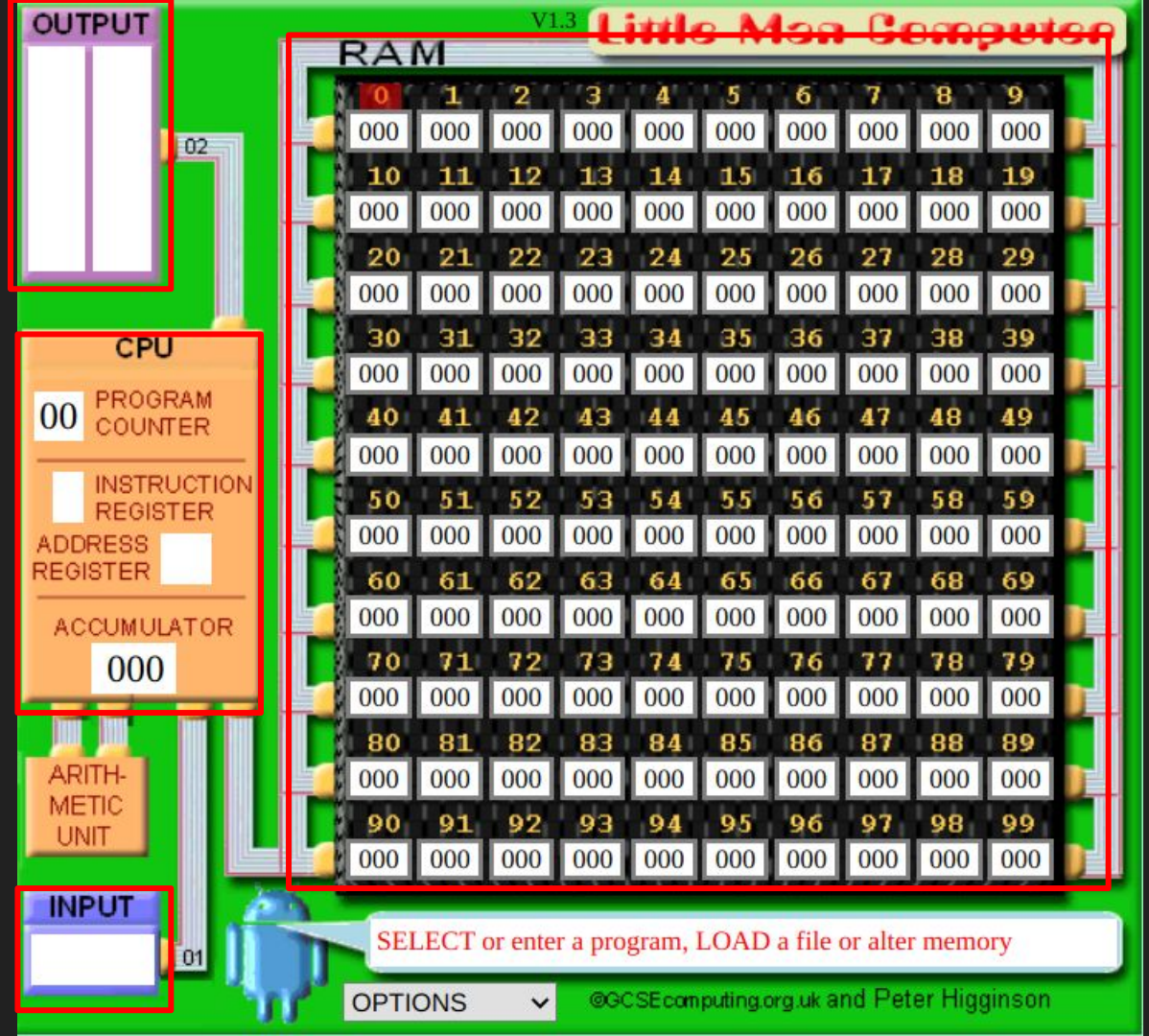
- Đọc qua tài liệu buổi học trước khi tới lớp sẽ giúp bạn chuẩn bị tốt hơn
- Thử google 1 lần trước khi đặt câu hỏi sẽ giúp bạn học cách tìm kiếm và cũng tiết kiệm thời gian hơn
- Càng làm bài tập đầy đủ và đọc tài liệu thì các bạn sẽ càng học được nhiều sau khóa học này

Agenda

1. Introduction
   a. Giới thiệu về HSGS Sec
   b. Nội dung khóa học
   c. Nội quy học tập
2. Giới thiệu về assembly language (asm)
3. Các phép tính toán cơ bản với asm
   a. Registers
   b. add, sub
   c. mul, div, mod

Little Man Computer Revisited

- Little Man Computer - CPU simulator
- Cấu trúc cơ bản của máy tính
  - Input/Output
  - CPU
  - Bộ nhớ/RAM


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

Little Man Computer Revisited

- Về cơ bản, máy tính chỉ hiểu được những chỉ dẫn vô cùng đơn giản
- Những chỉ dẫn này được biểu diễn bằng assembly language (hợp ngữ) để con người có thể đọc hiểu

Agenda

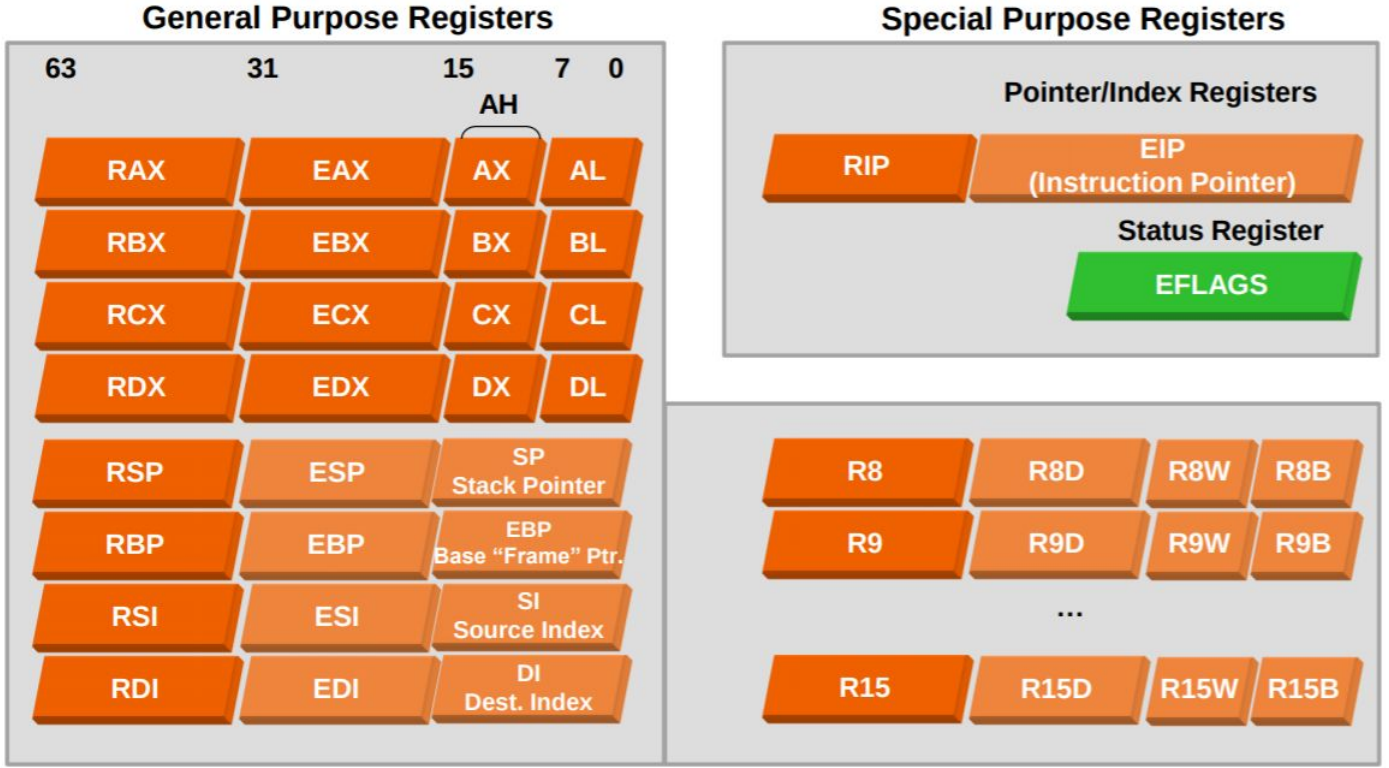
1. Introduction
    a. Giới thiệu về HSGS Sec
    b. Nội dung khóa học
    c. Nội quy học tập
2. Giới thiệu về assembly language (asm)
3. Các phép tính toán cơ bản với asm
    a. Registers
    b. add, sub
    c. mul, div, mod

Registers - Biến của CPU

- Để có thể tính toán và làm các công việc khác, CPU cũng cần sử dụng biến
- Biến của CPU có tên gọi là register - thanh ghi
- Các register được sử dụng thường xuyên trong khóa học: rax, rbx, rcx, rdx, rdi, rsi
- Mỗi register trên có độ lớn là 64-bit (8-byte). Đây là độ dài tối đa của register trên máy tính 64-bit. Ngoài ra còn có các register con của các register trên
- Register không có kiểu, chỉ chứa dữ liệu

Intel (IA-32/64) Architectures

CS:APP 3.4



Sơ lược về các chỉ dẫn trong assembly

- Mỗi một dòng code thường có cấu trúc như sau

  <operator> <destination>, <source>

  <operator> <operand 1> [,<operand 2>, ..., <operand N>]

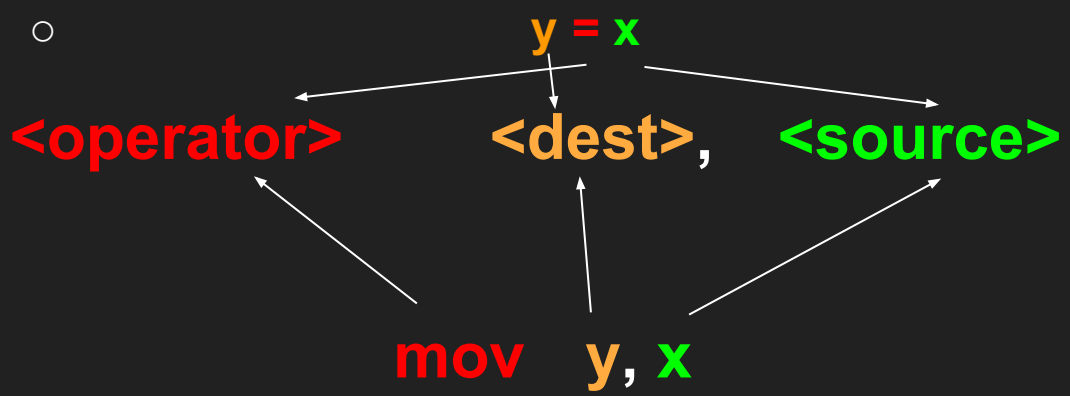
- Destination và Source có thể là các register, hoặc số, hoặc địa chỉ bộ nhớ. Điều này tùy thuộc vào Operator là gì.
- Ví dụ sẽ có ngay sau đây :D

add/sub/mov

- Thực hiện phép gán **x = x + 1** như nào trong assembly?

add/sub/mov

- Thực hiện phép gán **y = x + 1** như nào trong assembly?
- Không thể thực hiện trực tiếp phép tính này được, buộc phải thực hiện qua 2 bước
  - x = x + 1
  -



add/sub/mov

- Thực hiện phép gán **rax = rax - 1** như nào trong assembly?

mul/div/mod

- rax = rax * rbx ⇒ imul rax, rbx
- rax = rbx * 31 ⇒ imul rax, rbx, 31
- Điều kiện thực hiện phép nhân là kết quả có thể chứa hoàn toàn trong register <dest> (64-bit)
- Nếu kết quả phép tính vượt ngoài độ lớn 64-bit thì register <dest> sẽ chữa 64-bit bé nhất của kết quả.

mul/div/mod

- Assembly cũng hỗ trợ phép nhân có kết quả vượt ngoài độ lớn của 1 register
- rdx:rax = rax * rbx ⇒ imul rbx
- Ở cách dùng instruction này, <dest> là register cố định đã được chỉ định sẵn, và 1 <operand> đã được chỉ định sẵn là register rax
- 64-bit bé nhất của kết quả được chứa tại rax
- 64-bit lớn nhất của kết quả được chứa tại rdx

⇒ Đây là thao tác có 1 toán tử (1 operand)

mul/div/mod

- Phép chia có cách thức hoạt động gần tương tự phép nhân 1 toán tử
- rdx:rax = rax * rbx + rdx ⇒ rax = rdx:rax / rbx (mod rdx) ⇒ idiv rbx
- rdx:rax là số bị chia, toán tử <operand> là số chia
- Sau khi thực hiện phép tính, rax chứa thương, rdx chứa số dư
- **Chú ý: Nếu thương có kết quả lớn hơn 64-bit thì sẽ không chứa được trong 1 register. Lúc này sẽ xảy ra lỗi.**

mul/div/mod

- Tuỳ thuộc vào độ lớn của các register toán tử mà phép tính có thể lưu kết quả ở vị trí khác nhau
- Cùng 1 operator có thể có nhiều cách dùng khác
- Đọc thêm tài liệu [3] và [4]

OK cùng code nào :D

Bài Tập Về Nhà

1. Sử dụng phần tài liệu tham khảo và các nguồn khác, tìm hiểu về các phép toán trên bit (and, or, xor, not, shift left, shift right, etc.), các cách dùng operator khác nhau (1, 2, 3 toán tử).
2. Bài tập lập trình: Hoàn thành các bài tập thuộc buổi 01 trên kjudge tại địa chỉ sau

Tài liệu tham khảo

- [1] Guide to x86 Assembly
- [2] Assembly Programming Tutorial
- [3] IDIV — Signed Divide
- [4] IMUL — Signed Multiply
- [5] Assembly x86 Emulator
- [6] x86 and amd64 instruction reference