Writing an arm assembly

- Introduction to ARM assembly basics
  - If you don't have ARM device, you can setup your own lab environment in Virtual Machine using QEMU and raspberry pi distro
    - https://azeria-labs.com/emulate-raspberry-pi-with-qemu/
  - Debuging with GDB
    - การ save file & compile
      - $as [name.s] -o [name.o]
        $ ld [name.o] -o [name]
    - debugger can → load a memory dump after a crash (post-mortem debugging)
      → attach to a running process (used for server process)
      → launch a program and debug it
    - ติดตั้งเสริม GEF
      - การใช้การช่วยเหลือ (gdb) h
        (gdb) apropos [search-item]
    - Breakpoint command
      - break (or just b) <*function-name*>
      - break <*line-number*>
      - break *filename:function*
      - break *filename:line-number*
      - break *<*address*>
      - break +<*offset*>
      - break –<*offset*>
      - tbreak (set a temporary breakpoint)
      - del <*number*> (delete breakpoint number x)
      - delete (delete all breakpoints)
      - delete <*range*> (delete breakpoint ranges)
      - disable/enable <*breakpoint-number-or-range*> (does not delete breakpoints, just enables/disables them)
      - continue (or just c) – (continue executing until next breakpoint)
      - continue <*number*> (continue but ignore current breakpoint *number* times. Useful for breakpoints within a loop.)
      - finish (continue to end of function)
    - การ run command
      - gef> run
    - to quit the programe
      gef> q
    - to stop execution
      gef> ctrl
    - to display memory content
      gef> x

↳ to display memory content

Syntax: x/<count><format><unit>	
FORMAT	UNIT
x – Hexadecimal	b – bytes
d – decimal	h – half words (2 bytes)
i – instructions	w – words (4 bytes)
t – binary (two)	g – giant words (8 bytes)
o – octal	
u – unsigned	
s – string	
c – character	

↳ Commands for stepping through the code.

Commands for stepping through the code:

- Step to next line of code. Will step into a function
  - stepi
  - s
  - step <number-of-steps-to-perform>
- Execute next line of code. Will not enter functions
  - nexti
  - n
  - next <number>
- Continue processing until you reach a specified line number, function name, address, filename:function, or filename:line-number
  - until
  - until <line-number>
- Show current line number and which function you are in
  - where

```
gef> nexti 5
...
```

เพิ่มเติมได้ในลิงก์นี้เลยครับ

https://azeria-labs.com/debugging-with-gdb-introduction/

- ARM vs INTEL processor
  - main difference
    - instruction set
      - Intel is CISC (complex instruction set computing)
        - processor that has more and larger feature-rich instruction set
          - allow many complex instructions
          - less register than ARM
      - ARM is RISC (reduced instruction set computing) ⟶ มี 2 mode ARM, Thumb mode
        - processor that has a simplified instruction (≃ 100 instructions)
        - ARM's instructions operate only on registe and uses Load/Store memory model for memory access. → มีแค่ Load/Store เท่านั้นที่สามารถเข้าถึง memory ได้นั่นเอง
          - หลักๆ 3 รูปแบบ → load, increment, store
          - first load the value at the particular address into a register, increment it with in the register and store it back to the memory from the register.
      - ความเรียบง่ายของมันก็เป็นข้อเสีย และก็ข้อดี
        - ข้อดี
          - ไว
        - ข้อเสีย
          - the less instructions means a greater emphasis
  - ARM มีความแตกต่างกันในแต่ละรุ่นของมันเลย แต่ก็มีความเป็น generic → เมื่อเข้าใจพื้นฐาน ก็ไปต่อได้หมด
- writing assembly
  - If we want to do Reverse Engineering and understand the program flow of ARM binaries, build your own ARM shellcode, craft ARM ROP chains and debug ARM app.
- What exactly Assembly language?
  - ภาษาที่เหนือภาษาเครื่องขึ้นมาอีกหนึ่ง → คนจะมองเข้าใจได้ง่าย → ไม่ต้องไปเขียน 0101 แบบนี้ จ่อรอตาย?
  - ต้องใช้ GNU Assembler from GNU Binutills → working with *.s extension
  - as [name.s] -o [name.o] → as เป็น tool ของ Binutills
    ld [name.o] -o [name]
  - 1110 0001 1010 0000 0010 0000 0000 0001
    - เราไม่สามารถจำได้ว่า pattern ไหนหมายถึงอะไร → "mnemonics" มาช่วยในการจำรูปแบบ binary
      - set of mnemonics คือ Assembly language program

↳ เผื่อมีความสนใจเพิ่มเติม

- ↳ https://www.coranac.com/tonc/text/asm.htm
- ↳ http://thinkingeek.com/arm-assembler-raspberry-pi/
- ↳ http://infocenter.arm.com/help/topic/com.arm.doc.dui0068b/index.html
- ↳ http://www.keil.com/support/man/docs/armasm/default.htm

↳ DATA Type

↳ similar to high level languages, ARM supports operations on different datatypes.

↳ the extension for these data type

↳ -h, -sh = halfwords
-b, -sb = bytes
no extension for word

↳ เมื่อเรามาลองเทียบ ldr = Load Word
ldrh = load unsigned half word
ldrsh = load singed half word

↳ ตามที่เก็บข้อมูลเป็นแบบ little-endian (LE) ไม่ก็ Big-endian (BE)
↳ least อยู่ lowest address ↳ most sig bit อยู่ lowest address

↳ ARM registers

↳ amount of register depend on the ARM version

↳ ตัวอย่างดูว่า register ไหนแต่ละตัวมีใช้งานยังไงบ้าง

↳ หลักๆ มีและถูกแบ่งเป็น 2 groups

#	Alias	Purpose
R0	–	General purpose
R1	–	General purpose
R2	–	General purpose
R3	–	General purpose
R4	–	General purpose
R5	–	General purpose
R6	–	General purpose
R7	–	Holds Syscall Number
R8	–	General purpose
R9	–	General purpose
R10	–	General purpose
R11	FP	Frame Pointer
Special Purpose Registers		
R12	IP	Intra Procedural Call
R13	SP	Stack Pointer
R14	LR	Link Register
R15	PC	Program Counter
CPSR	–	Current Program Status Register

ประมาณนี้

แล้วแต่รุ่นด้วยนะ

ตารางเทียบค่าการเทียบกับ intel

ARM	Description	x86
R0	General Purpose	EAX
R1-R5	General Purpose	EBX, ECX, EDX, ESI, EDI
R6-R10	General Purpose	–
R11 (FP)	Frame Pointer	EBP
R12	Intra Procedural Call	–
R13 (SP)	Stack Pointer	ESP
R14 (LR)	Link Register	–
R15 (PC)	<- Program Counter / Instruction Pointer ->	EIP
CPSR	Current Program State Register/Flags	EFLAGS

Path อธิบายการทำงาน หรือหน้าที่ของ register ณ ตำแหน่งต่างๆ