CS143A
Principles on Operating Systems
Discussion 03:

Instructor: Prof. Anton Burtsev

TA: Saehanseul Yi (Hans)

Oct 18, 2019 **3PM**

Just a draft

http://bit.ly/2MpYDKr

Agenda

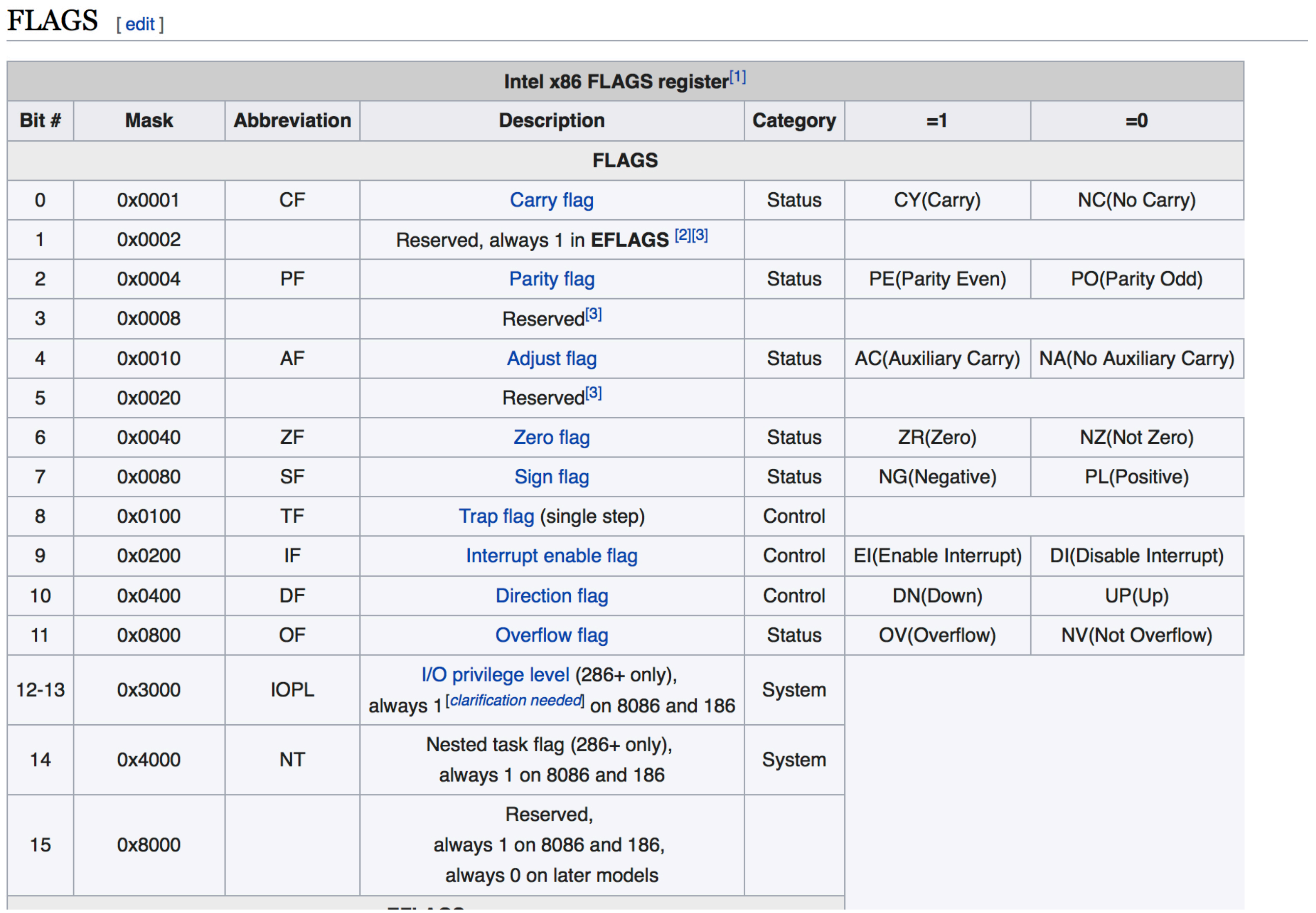
- HW1 part 5 review & walk-through
- (optional) gdb-dashboard

Instructions

- IA-32/IA-64 Software Developer’s Manual
- https://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-instruction-set-reference-manual-325383.pdf

Instructions: EFLAGS

- https://en.wikipedia.org/wiki/FLAGS_register



FLAGS [edit]

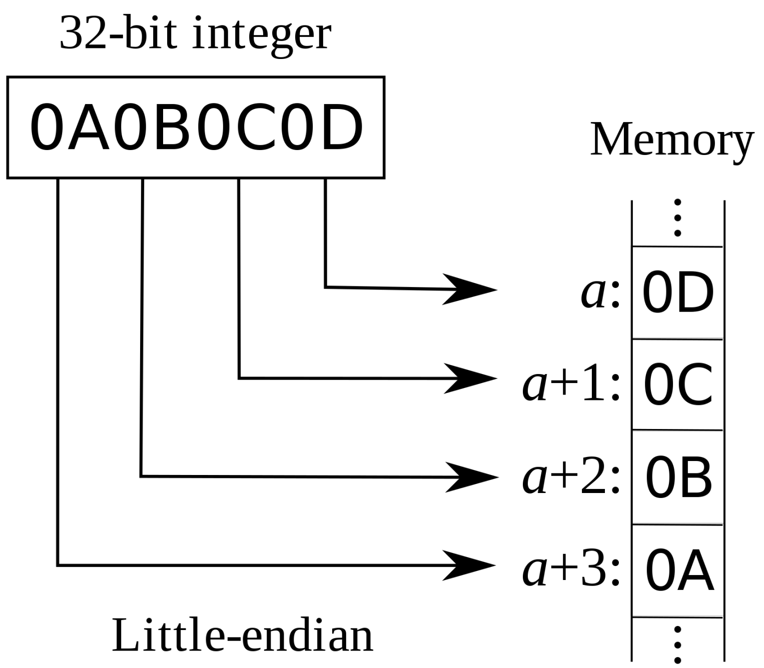
Bit #	Mask	Abbreviation	Description	Category	=1	=0
Intel x86 FLAGS register[1]						
FLAGS						
0	0x0001	CF	Carry flag	Status	CY(Carry)	NC(No Carry)
1	0x0002		Reserved, always 1 in **EFLAGS** [2][3]			
2	0x0004	PF	Parity flag	Status	PE(Parity Even)	PO(Parity Odd)
3	0x0008		Reserved[3]			
4	0x0010	AF	Adjust flag	Status	AC(Auxiliary Carry)	NA(No Auxiliary Carry)
5	0x0020		Reserved[3]			
6	0x0040	ZF	Zero flag	Status	ZR(Zero)	NZ(Not Zero)
7	0x0080	SF	Sign flag	Status	NG(Negative)	PL(Positive)
8	0x0100	TF	Trap flag (single step)	Control		
9	0x0200	IF	Interrupt enable flag	Control	EI(Enable Interrupt)	DI(Disable Interrupt)
10	0x0400	DF	Direction flag	Control	DN(Down)	UP(Up)
11	0x0800	OF	Overflow flag	Status	OV(Overflow)	NV(Not Overflow)
12-13	0x3000	IOPL	I/O privilege level (286+ only), always 1[clarification needed] on 8086 and 186	System		
14	0x4000	NT	Nested task flag (286+ only), always 1 on 8086 and 186	System		
15	0x8000		Reserved, always 1 on 8086 and 186, always 0 on later models			

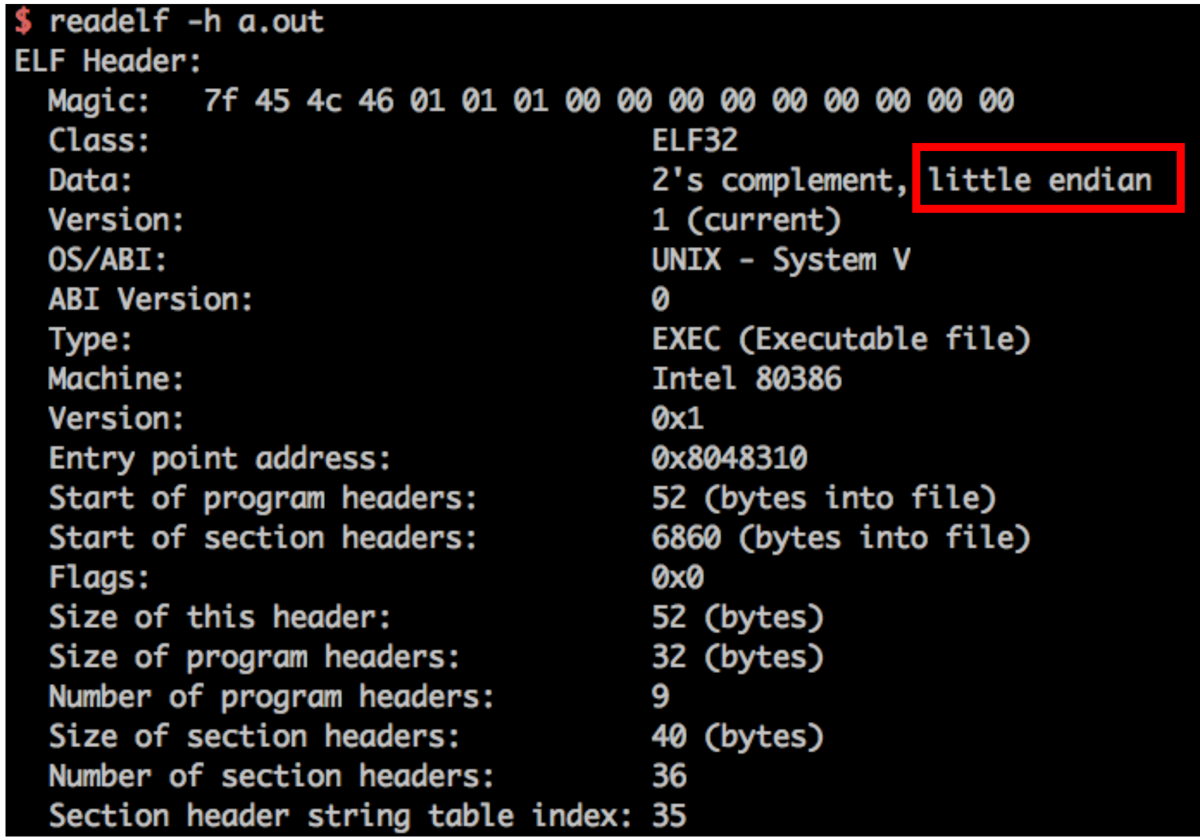
Difference between AT&T and Intel Syntax

- In ICS 143A, **we use Intel syntax**
- AT&T immediate operands use a $ to denote them, whereas Intel immediate operands are undelimited
- AT&T prefaces register names with a %, while Intel does not
- AT&T syntax uses the opposite order for source and destination operands.
- ...
- http://shawnleezx.github.io/blog/2013/12/11/main-difference-between-intel-and-at-and-t-syntax-assembly-language/

Endianness: Little vs. Big

- https://en.wikipedia.org/wiki/Endianness





```
$ readelf -h a.out
ELF Header:
  Magic:   7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF32
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              EXEC (Executable file)
  Machine:                           Intel 80386
  Version:                           0x1
  Entry point address:               0x8048310
  Start of program headers:          52 (bytes into file)
  Start of section headers:          6860 (bytes into file)
  Flags:                             0x0
  Size of this header:               52 (bytes)
  Size of program headers:           32 (bytes)
  Number of program headers:         9
  Size of section headers:           40 (bytes)
  Number of section headers:         36
  Section header string table index: 35
```

Recap

- Caller pushes arguments to stack
- ‘call’ instruction pushes the return address in stack
- Callee saves old ebp into stack
- Local variables are stored in stack
- Callee makes room for local vars by subtracting from stack pointer
- Registers EAX, ECX, and EDX are caller-saved, and the rest are callee-saved
- EAX is reserved for the return value
- Before returning: restore the old ebp from stack as well as esp

Before we start..

- Intel Software’s Manual
- DEC-HEX converter
- Scratch pad

HW1 Part 5

- Start GDB and set the breakpoint on the **sum** function, and **run** the program
- Use the x command to inspect the stack
- Explain every value from the dump that you get

```
unsigned long sum(int n) {
    int i;
    unsigned long sum = 0;

    for (i = 0; i < n; i++) {
        sum = sum + i;
    }

    return sum;
}
```

```
int main(void) {

    unsigned long s;

    s = sum(100);
    printf("Hello world, the sum:%ld\n", s);
    return 0;
}
```

Scratch pad

- address of next instruction after sum(): 0x0804844f
- stack pointer(esp) before sum():0xffffc4a0
- frame pointer(ebp) before sum(): 0xffffc4c8

Stack memory when entering sum()

Address	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0xffffc49c	4f	84	04	08	64	00	00	00	64	c5	ff	ff	6c	c5	ff	ff
0xffffc4ac	bd	39	e1	f7	c4	83	fa	f7	00	80	00	00	7b	84	04	08
0xffffc4bc	00	80	fa	f7	70	84	04	08	00	00	00	00	00	00	00	00
0xffffc4cc	a3	b2	df	f7	01	00	00	00	64	c5	ff	ff	6c	c5	ff	ff
0xffffc4dc	b0	86	fd	f7	01	00	00	00	01	00	00	00	00	00	00	00
0xffffc4ec	10	a0	04	08	1c	82	04	08	00	80	fa	f7	00	00	00	00

Return address of the sum

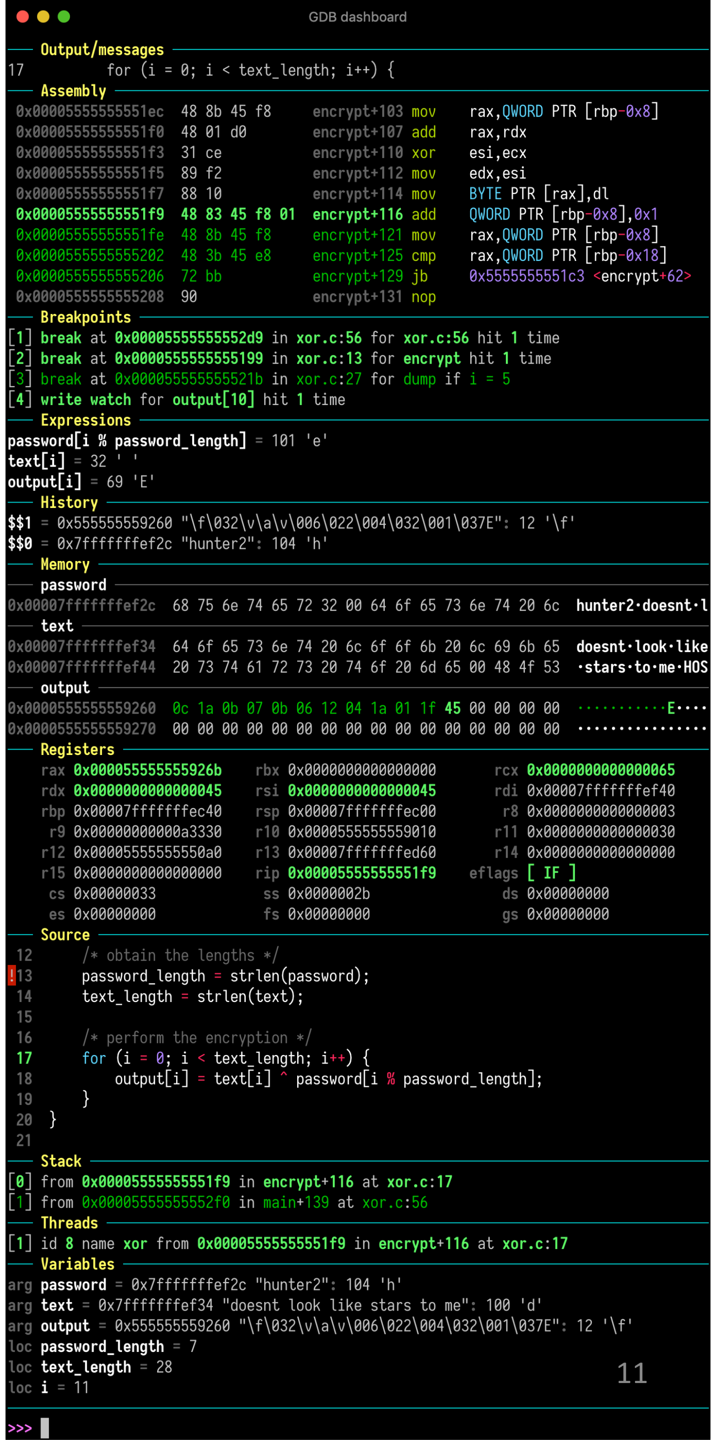
The argument of sum()

main()'s stack(local variables, ….)

old ebp for main

GDB-dashboard

- https://github.com/cyrus-and/gdb-dashboard
- Highlighting & coloring gdb outputs
- reduce the number of gdb commands needed to inspect the program
- More options: https://stackoverflow.com/questions/209534/how-to-highlight-and-color-gdb-output-during-interactive-debugging/17341335#17341335

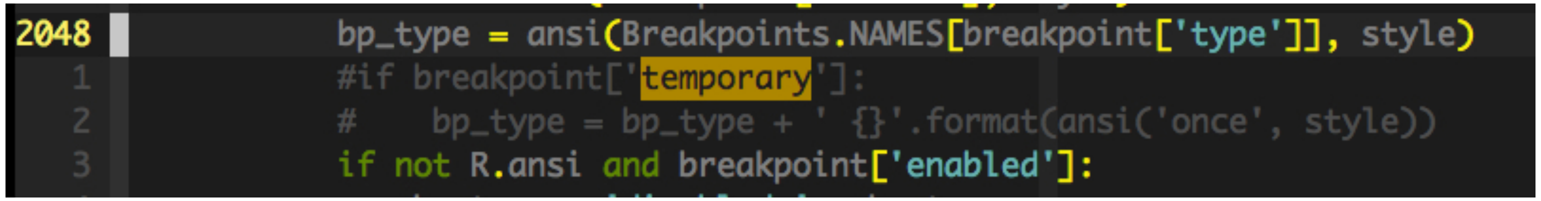


GDB dashboard

```
─── Output/messages ──────────────────────────────────────────
17          for (i = 0; i < text_length; i++) {
─── Assembly ─────────────────────────────────────────────────
 0x00005555555551ec  48 8b 45 f8     encrypt+103 mov    rax,QWORD PTR [rbp-0x8]
 0x00005555555551f0  48 01 d0        encrypt+107 add    rax,rdx
 0x00005555555551f3  31 ce           encrypt+110 xor    esi,ecx
 0x00005555555551f5  89 f2           encrypt+112 mov    edx,esi
 0x00005555555551f7  88 10           encrypt+114 mov    BYTE PTR [rax],dl
 0x00005555555551f9  48 83 45 f8 01  encrypt+116 add    QWORD PTR [rbp-0x8],0x1
 0x00005555555551fe  48 8b 45 f8     encrypt+121 mov    rax,QWORD PTR [rbp-0x8]
 0x0000555555555202  48 3b 45 e8     encrypt+125 cmp    rax,QWORD PTR [rbp-0x18]
 0x0000555555555206  72 bb           encrypt+129 jb     0x5555555551c3 <encrypt+62>
 0x0000555555555208  90              encrypt+131 nop
─── Breakpoints ──────────────────────────────────────────────
[1] break at 0x00005555555552d9 in xor.c:56 for xor.c:56 hit 1 time
[2] break at 0x0000555555555199 in xor.c:13 for encrypt hit 1 time
[3] break at 0x000055555555521b in xor.c:27 for dump if i = 5
[4] write watch for output[10] hit 1 time
─── Expressions ──────────────────────────────────────────────
password[i % password_length] = 101 'e'
text[i] = 32 ' '
output[i] = 69 'E'
─── History ──────────────────────────────────────────────────
$$1 = 0x555555559260 "\f\032\v\a\v\006\022\004\032\001\037E": 12 '\f'
$$0 = 0x7fffffffef2c "hunter2": 104 'h'
─── Memory ───────────────────────────────────────────────────
─── password ─────────────────────────────────────────────────
0x00007fffffffef2c  68 75 6e 74 65 72 32 00 64 6f 65 73 6e 74 20 6c  hunter2·doesnt·l
─── text ─────────────────────────────────────────────────────
0x00007fffffffef34  64 6f 65 73 6e 74 20 6c 6f 6f 6b 20 6c 69 6b 65  doesnt·look·like
0x00007fffffffef44  20 73 74 61 72 73 20 74 6f 20 6d 65 00 48 4f 53  ·stars·to·me·HOS
─── output ───────────────────────────────────────────────────
0x0000555555559260  0c 1a 0b 07 0b 06 12 04 1a 01 1f 45 00 00 00 00  ···········E····
0x0000555555559270  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ················
─── Registers ────────────────────────────────────────────────
    rax 0x000055555555926b    rbx 0x0000000000000000       rcx 0x0000000000000065
    rdx 0x0000000000000045    rsi 0x0000000000000045       rdi 0x00007fffffffef40
    rbp 0x00007fffffffec40    rsp 0x00007fffffffec00        r8 0x0000000000000003
     r9 0x00000000000a3330    r10 0x0000555555559010       r11 0x0000000000000030
    r12 0x00005555555550a0    r13 0x00007fffffffed60       r14 0x0000000000000000
    r15 0x0000000000000000    rip 0x00005555555551f9    eflags [ IF ]
     cs 0x00000033             ss 0x0000002b                ds 0x00000000
     es 0x00000000             fs 0x00000000                gs 0x00000000
─── Source ───────────────────────────────────────────────────
 12      /* obtain the lengths */
!13      password_length = strlen(password);
 14      text_length = strlen(text);
 15
 16      /* perform the encryption */
 17      for (i = 0; i < text_length; i++) {
 18          output[i] = text[i] ^ password[i % password_length];
 19      }
 20  }
 21
─── Stack ────────────────────────────────────────────────────
[0] from 0x00005555555551f9 in encrypt+116 at xor.c:17
[1] from 0x00005555555552f0 in main+139 at xor.c:56
─── Threads ──────────────────────────────────────────────────
[1] id 8 name xor from 0x00005555555551f9 in encrypt+116 at xor.c:17
─── Variables ────────────────────────────────────────────────
arg password = 0x7fffffffef2c "hunter2": 104 'h'
arg text = 0x7fffffffef34 "doesnt look like stars to me": 100 'd'
arg output = 0x555555559260 "\f\032\v\a\v\006\022\004\032\001\037E": 12 '\f'
loc password_length = 7
loc text_length = 28
loc i = 11
──────────────────────────────────────────────────────────────
>>> 
```

GDB-dashboard: Install and Patch

- wget -P ~ https://git.io/.gdbinit
  pip install pygments –user
- **AttributeError: 'module' object has no attribute 'COMPLETE_EXPRESSION'**
  - mkdir ~/.gdbinit.d
  - echo "gdb.COMPLETE_EXPRESSION = gdb.COMPLETE_SYMBOL" > ~/.gdbinit.d/COMPLETE_EXPRESSION.py
- **AttributeError: 'gdb.Breakpoint' object has no attribute 'temporary'**
  - comment out all the 'temporary' in ~/.gdbinit (line 327, 2049-2050)
  - put # in front the line



```
2048            bp_type = ansi(Breakpoints.NAMES[breakpoint['type']], style)
   1            #if breakpoint['temporary']:
   2            #    bp_type = bp_type + ' {}'.format(ansi('once', style))
   3            if not R.ansi and breakpoint['enabled']:
```

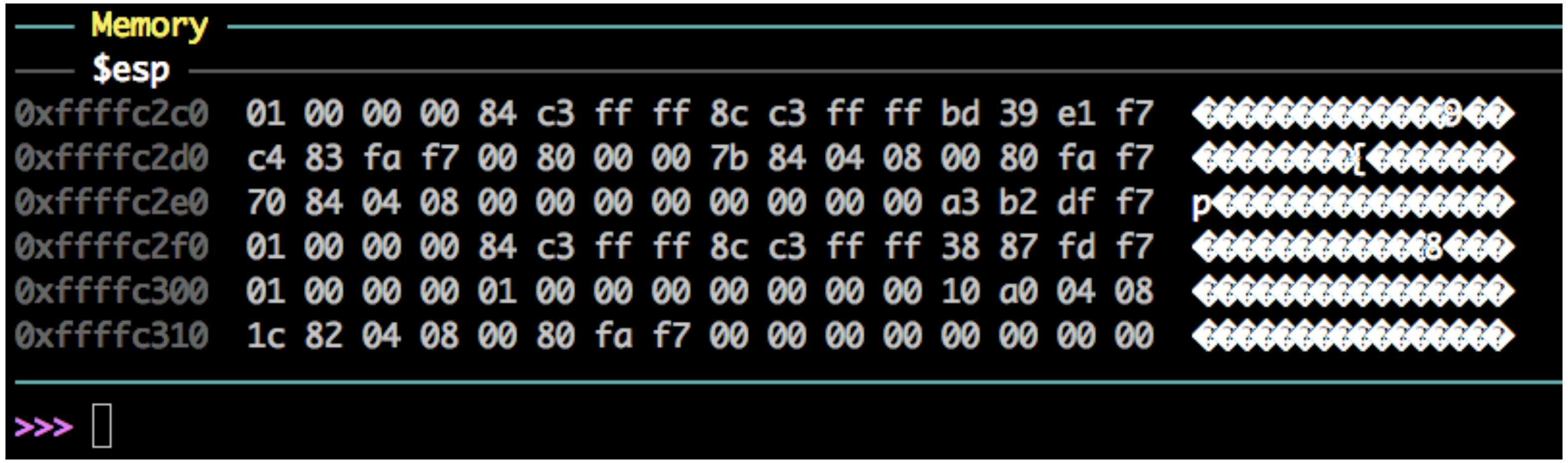
- More errors… https://github.com/cyrus-and/gdb-dashboard/wiki/Support-older-GDB-versions

GDB-dashboard: layout

- dashboard -layout assembly breakpoints expressions history memory registers source stack threads variables
  - **expressions**: Watch user expressions
  - **history**: List the last entries of the value history.
  - **stack**: Call stack(<u>NOT the stack memory</u>)
- dashboard -layout source assembly registers memory
  - my setting for this discussion section

GDB dashboard: Monitor stack memory

- dashboard memory watch $esp 24*4



```
─── Memory ──────────────────────────────────────────────────────────────
─── $esp ────────────────────────────────────────────────────────────────
0xffffc2c0  01 00 00 00 84 c3 ff ff 8c c3 ff ff bd 39 e1 f7  �������������9��
0xffffc2d0  c4 83 fa f7 00 80 00 00 7b 84 04 08 00 80 fa f7  ��������{�������
0xffffc2e0  70 84 04 08 00 00 00 00 00 00 00 00 a3 b2 df f7  p���������������
0xffffc2f0  01 00 00 00 84 c3 ff ff 8c c3 ff ff 38 87 fd f7  ������������8���
0xffffc300  01 00 00 00 01 00 00 00 00 00 00 00 10 a0 04 08  ����������������
0xffffc310  1c 82 04 08 00 80 fa f7 00 00 00 00 00 00 00 00  ����������������
─────────────────────────────────────────────────────────────────────────
>>> 
```