Systems@ETH Zürich

Exercise Session 4

Computer Architecture and Systems Programming

Herbstsemester 2016

Agenda

- Review of Exercise 3
- Assembly
- Outlook to Exercise 4
- Quiz

Assignment 3

Assignment 3

- Same story as last time:

  – Check for ptr != null before dereferencing it.

Assignment 3

- ComplexSet:

```
struct complex_set {
   int size;
   struct complex *points;
};
```

```
struct complex_set *cset_alloc(struct complex c_arr[], int size)
{
    assert(size > 0);
    ComplexSet *newset;
    newset = malloc(sizeof(*newset));
    if (!newset) return NULL;
    newset->size = size;
    newset->points = malloc(size * sizeof(struct complex));
    if (!newset->points) { free(newset); return NULL; }
    memcpy(newset->points, c_arr, size * sizeof(struct complex));
    return newset;
}
```

Assignment 3

- ComplexSet:

```
struct complex_set {
   int size;
   struct complex *points;
};
```

```
void cset_free(struct complex_set *set) {
    if (!set) return;
    if (set->points) { free(set->points); }
    free(set);
    return;
}
```

Assignment 3

- File I/O

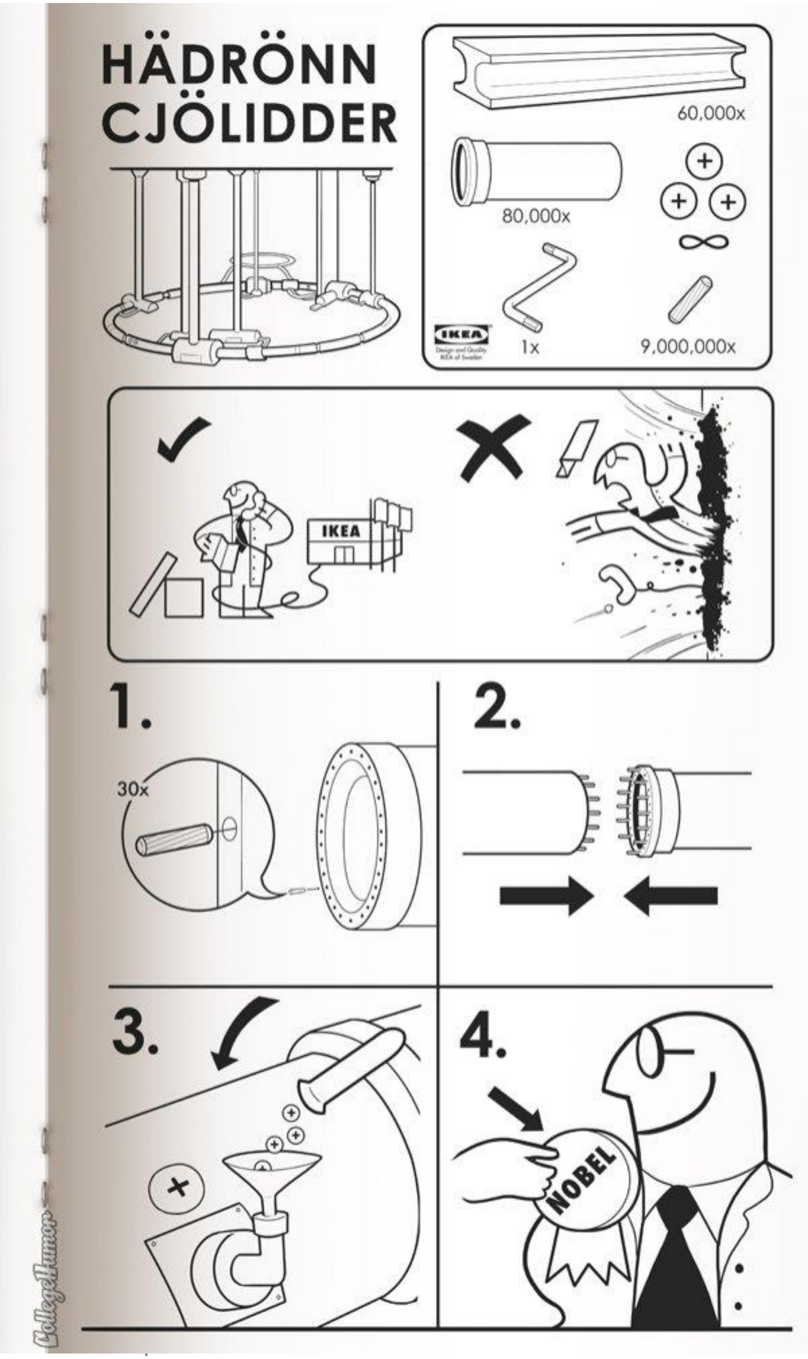
```c
while(fgets(line, STRSIZE*NFIELDS, fp)) {
    /*parse the fields*/
    fields_read = sscanf(line,"%s %s %s %s %s %s %d %d %d",
                       state_code_org, country_code_org,
                       state_code_dest, country_code_dest,
                       state_abbrv, state_name, &return_num,
                       &exmpt_num,  &aggr_agi);

    if(strcmp(state_code_org, "\"25\"") == 0) {
        printf("%-30s, %6d\n", state_name, aggr_agi);
        total += aggr_agi;
    }
}
```

Assignment 3 discussion

wc:

```
enum state { INSIDE, OUTSIDE };
enum state currstate = OUTSIDE;
while ((c = getc(fp)) != EOF) {
    nc++;
    if (c == '\n') { nl++; }
    if (isspace(c)) {
        if (currstate = INSIDE) { nw++; }
        currstate = OUTSIDE;
    } else {
        currstate = INSIDE;
    }
}
```

Assembly



- Not this assembly…

Obtaining Assembly

- You can have GCC to output assembly code

```
gcc -S code.c
```

- This will give you code.s

- Some remarks
  - Be careful with optimization flags –O
  - You can even compile single C-files without main()

-Ox: The Effects on the Code

```
/* string.c */

char string_init(void) {
  char s[] = "Hello";
  return s[1];
}
```

```
/* -O0 */

string_init:
        pushq   %rbp
        movq    %rsp, %rbp
        movl    $1819043144, -16(%rbp)
        movw    $111, -12(%rbp)
        movzbl -15(%rbp), %eax
        popq    %rbp
        ret
```

-Ox: The Effects on the Code

```
/* string.c */

char string_init(void) {
  char s[] = "Hello";
  return s[1];
}
```

```
/* -O0 */

string_init:
        pushq  %rbp
        movq   %rsp, %rbp
        movl   $1819043144, -16(%rbp)
        movw   $111, -12(%rbp)
        movzbl -15(%rbp), %eax
        popq   %rbp
        ret
```

```
/* -O */

string_init:
  movl $101, %eax
  ret
```

x86-64 integer registers

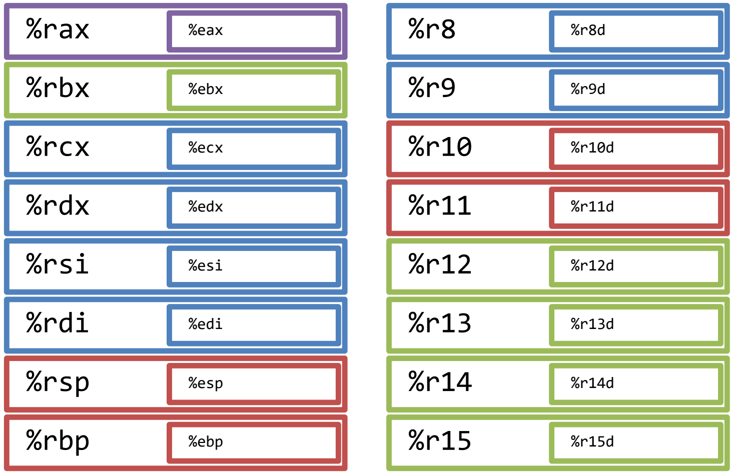
general purpose	%rax	%eax	%r8	%r8d
	%rbx	%ebx	%r9	%r9d
	%rcx	%ecx	%r10	%r10d
	%rdx	%edx	%r11	%r11d
	%rsi	%esi	%r12	%r12d
	%rdi	%edi	%r13	%r13d
	%rsp	%esp	%r14	%r14d
	%rbp	%ebp	%r15	%r15d
	%rip	%eip	%rsr	%esr

Moving data

- `movx` *Source, Dest*
  - x in {b, w, l,q}

  - `movq` *Source, Dest*:
    Move 8-byte "quad word"
  - `movl` *Source, Dest*:
    Move 4-byte "long word"
  - `movw` *Source, Dest*:
    Move 2-byte "word"
  - `movb` *Source, Dest*:
    Move 1-byte "byte"

- Lots of these in typical code

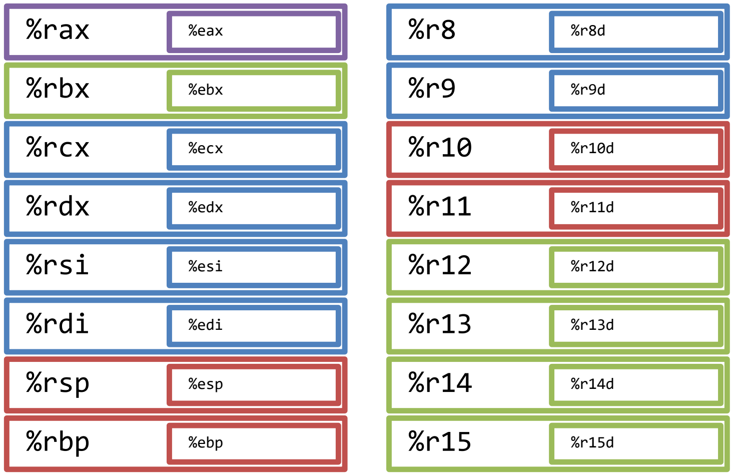


%rax	%eax	%r8	%r8d
%rbx	%ebx	%r9	%r9d
%rcx	%ecx	%r10	%r10d
%rdx	%edx	%r11	%r11d
%rsi	%esi	%r12	%r12d
%rdi	%edi	%r13	%r13d
%rsp	%esp	%r14	%r14d
%rbp	%ebp	%r15	%r15d

Moving data

`movx` *Source*, *Dest*:

- Operand Types
  - ***Immediate:*** Constant integer data
    - Example: `$0x400`, `$-533`
    - Like C constant, but prefixed with `'$'`
    - Encoded with 1, 2, 4, 8 bytes
  - ***Register:*** One of 16 integer registers
    - Example: `%eax`, `%r14d`
    - Note some (e.g. `%rsp`, `%rbp`) reserved for special use
    - Others have special uses for particular instructions
  - ***Memory:*** 1,2,4, or 8 consecutive bytes of memory at address given by register
    - Simplest example: `(%rax)`
    - Various other "address modes"



%rax	%eax	%r8	%r8d
%rbx	%ebx	%r9	%r9d
%rcx	%ecx	%r10	%r10d
%rdx	%edx	%r11	%r11d
%rsi	%esi	%r12	%r12d
%rdi	%edi	%r13	%r13d
%rsp	%esp	%r14	%r14d
%rbp	%ebp	%r15	%r15d

Complete memory addressing modes

- Most General Form:

D(Rb,Ri,S)	Mem[Reg[Rb]+S*Reg[Ri]+ D]

  - D: Constant “displacement” 1, 2, or 4 bytes (not 8!)
  - Rb: Base register: Any of 16 integer registers
  - Ri: Index register: Any, except for `%rsp`
    (Unlikely you’d use `%rbp`, either)
  - S: Scale: 1, 2, 4, or 8 (*why these numbers?*)

- Special Cases

(Rb,Ri)	Mem[Reg[Rb]+Reg[Ri]]
D(Rb,Ri)	Mem[Reg[Rb]+Reg[Ri]+D]
(Rb,Ri,S)	Mem[Reg[Rb]+S*Reg[Ri]]

Embedding Assembly into C

- **Problem:** Certain registers cannot by addressed by a variable in C directly

- **Observation:** You can access the registers via assembly instruction

- **Conclusion:** Embed assembly code into your C source file.

Inline Assembly

- Basic format to include inline assembly

```
__asm__("movb %bh (%eax)\n\t");
```

- Note: If the statement is unused, it may gets deleted!

```
__asm__ volatile ("movb %bh (%eax)\n\t");
```

- **Now:** how to get the contents of the register or provide data for the register?

Volatile ?

- The semantics of the volatile keyword differ from language to language

- C: “Do not optimize this away”
  Important when reading device registers
- Java: “Do read the value from the memory not from the cache. “
  Cf: Parallel Programming

http://en.wikipedia.org/wiki/Volatile_variable

Extended Inline Assembly

```
__asm__ ( assembler template
         : output operands /* optional */
         : input operands /* optional */
         : list of clobbered registers /* optional */
        );
```

These registers are modified, don’t store other values

```
int a=10, b;
__asm__ ("movl %1, %%eax; movl %%eax, %0;"
        :"=r"(b)
        :"r"(a)
        :"%eax"
        );
```

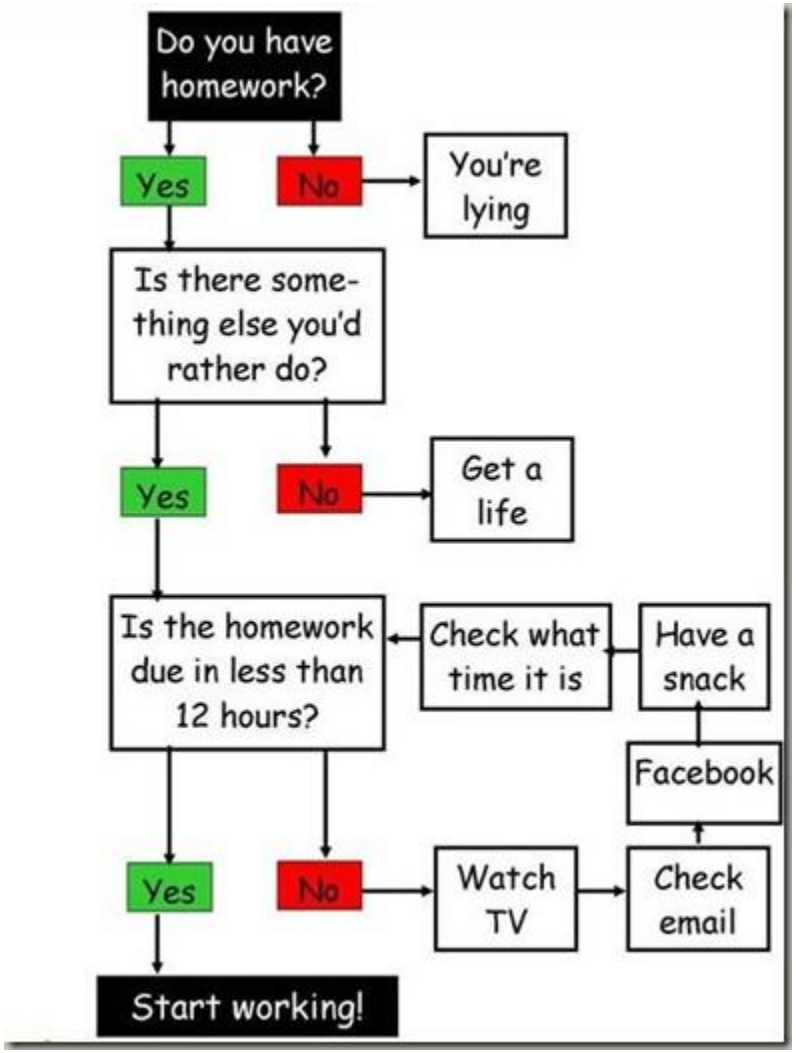
What’s happening here?

Assignment 4

Assignment 4

1. Implement a Hash-table
   - Input: char* -> Output: void*
   - Get more familiar with C pointers and memory management
   - How to hash strings? http://www.cse.yorku.ca/~oz/hash.html
2. x86 Assembly
   - Instructions (mov, lea, add, sub, dec, inc, mul)
   - Condition Codes

Submission



This time it’s mainly a pen-and-paper exercise.

Option 1: Send a PDF
Option 2: Hand in during exercise session

Quiz: Assembly

- Questions on Handout.

Quiz a)

- True
  - Tries to load address 0
- False
  - Computation on address
- False
  - Move the value 0 to %rcx
- False
  - Decreases the stack size

Quiz b)-d)

- b)
  - c) is the correct answer
- c)
  - a) Valid: %ebx = 4*%eax
  - b) Invalid: 15 is a memory address not intermediate!
  - c) Valid: store the content of %eax to address 655
- d)
  - a) is the correct answer
- e) mov %ebp, %esp

  pop %ebp

Quiz f)

C Code

```
// input: int x (in %ebx)
// output int y (in %eax)

int y = 0;
if (x > 0) {
  y = 10;
}
y += 5;
```

Assembly

```
xorl %eax, %eax
cmpl $0, %ebx
jle ELSE
movl $10, %eax
ELSE:
addl $5, %eax
```

```
xorl %eax, %eax
```

What is this doing?

Quiz g)

C Code

```c
// input: int x, int y
//        (in %ebx, %ecx)
// output: int z (in %eax)

int z = 0;
while (z <= y) {
  z += 3*(x+1);
}
```

Assembly

```
xorl %eax, %eax
leal 3(%ebx,%ebx,2), %edx
jmp CHECK
LOOP:
addl %edx, %eax
CHECK:
cmpl %ecx, %eax
jle LOOP
```

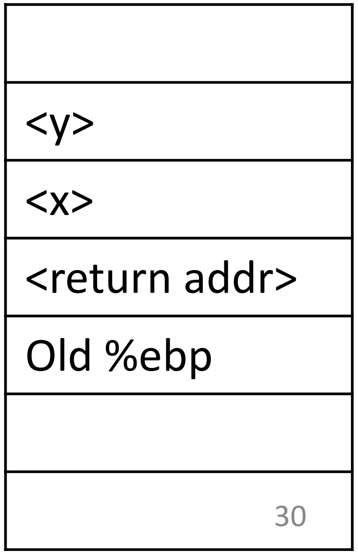
Quiz h)

Assembly

```
func:
    pushl  %ebp
    movl   %esp, %ebp
    movl   8(%ebp), %eax
    cmpl   12(%ebp), %eax
    jle    .L2
    movl   8(%ebp), %eax
    jmp    .L3
.L2:
    movl   12(%ebp), %eax
.L3:
    popl   %ebp
    ret
```

C Code

```
int func(int x, int y) {
  if (x > y) {
    return x;
  } else {
    return y;
  }
}
```



```
.def ___main; .scl 2; .type 32; .endef
.text
.align 32
LC0:
     .ascii "I will not Throw paper airplanes in class.\0"
     .globl _main
     .def _main; .scl 2; .type 32; .endef
     _main:
     pushl  %ebp
     movl  %esp, %ebp
     subl  $24, %esp
     andl  $-16, %esp
     movl  $0, %eax
     movl  %eax, -8(%ebp)
     movl  -8(%ebp), %eax
     call  __alloca
     call  ___main
     movl  $1, -4(%ebp)
L10:
    cmpl  $500, -4(%ebp)
    jle  L13
    jmp  L11
L13:
    movl  $LC0, (%esp)
    call  _printf
    leal  -4(%ebp), %eax
    incl  (%eax)
L11:
    movl  $0, %eax
    leave
    ret
```