CPS104 Recitation: Assembly Programming
Alexandru Duțu
Facts

- OS kernel and embedded software engineers use assembly for some parts of their code.
- Some operating systems had their entire GUIs written in assembly in the '80s.
Outline

- Assembly basics - review
- From C to assembly
- Team exercises
x86 instructions

- data transfer instructions
  - all variants of mov
  - push and pop

- binary arithmetic instructions
  - add, sub, imul, idiv, inc, dec, neg, cmp

- logical instructions
  - and, or, xor, nor

- shift-rotate instruction
  - sar, shr, ror, rol

- control instructions
  - jmp, je/jz, jne/jnz, ja/jnbe, jg/jnle, jge/jnl, call, ret

- misc
  - lea - load effective address
  - nop - no operation
Suffixes

- **b** - byte (8 bits)
- **s** - short (16 bits int or 32 bits float)
- **w** - word (16 bits)
- **l** - long (32 bits int or 64 bits float)
- **q** - quad (64 bits)
- **t** - ten bytes (80 bit floating point)
Prefixes

- % for registers
- $ for constants
Stack instructions

- push - pushes on the stack at %esp position
- pop - pops from the stack at %esp position
- call - function call
- ret - returns %eax as the result of the function
- enter, leave - set the base stack pointer before and after function execution
Function calls

- Arguments are push-ed on the stack
- EIP is push-ed also
- EBP (base pointer) is updated with the new ESP (stack pointer)
- Code of the function is executed
- EBP is restored to its previous value
- RET ends the function call (returns %eax)
Recommended framework for assembly

- C file that
  - has main()
  - sets the input
  - calls asm function
- assembly file (.S) that:
  - contains functions in assembly
/* main.c */
#include <stdio.h>

extern int sum(int x, int y);

int main(int argc, const char *argv[]) {
    int x, y, s, d, e;
    scanf("%d", &x);
    scanf("%d", &y);
    s = sum(x, y);
    printf("Sum: %d\n", s);
    return 0;
}

# func.S
.text
.globl sum

sum:
    pushl %ebp
    movl %esp, %ebp
    movl 8(%ebp), %eax
    movl 12(%ebp), %ebx
    addl %ebx, %eax
    leave
    ret

# save old base pointer
# set new base pointer
# get first argument
# get second argument
# sum y with x
# return value in %eax
Exercise

using the framework presented above, create one assembly function that takes 4 arguments $x, y, z, w$ and computes $x \cdot y + z \cdot w$
Solution

/* main.c */
#include <stdio.h>

extern int arithmetic(int x, int y, int w, int z);

int main()
{
    int x, y, z, w, e;
    scanf("%d", &x);
    scanf("%d", &y);
    scanf("%d", &z);
    scanf("%d", &w);

    s = arithmetic(x, y, w, z);
    printf("Sum: %d\n", s);

    return 0;
}

/* main.c */
#include <stdio.h>

extern int arithmetic(int x, int y, int w, int z);

int main()
{
    int x, y, z, w, e;
    scanf("%d", &x);
    scanf("%d", &y);
    scanf("%d", &z);
    scanf("%d", &w);

    s = arithmetic(x, y, w, z);
    printf("Sum: %d\n", s);

    return 0;
}
Compilation and debug

- `!gcc -g -m32 -o arithmetic main.c func.S`
- `!/<name_of_programm>`
- `-m32` flag is for compiling in 32 bit mode
- `:!gdb arithmetic`
  - `list main` - get line number of main function
  - `break <line number of main>` - put a breakpoint there
  - `run, next and step` - run
  - `p $eax ($ebx, $ecx...)` - print registers
if statement

- compare and jump instructions
- compare instructions
- cmpl
- jump instructions (branches)
- jmp, je, jne, jg, jng, jl, etc.

```python
if:
    comparison
    ...
    jg | jge | ... if_else
    #then body
    ...
    jmp if_end
if_else:
    #else body
    ...
    if_end:
```
e.g.:
.text
.globl sum

exp:
               # save old base pointer
    pushl %ebp
    movl %esp, %ebp  # set new base pointer
    movl 8(%ebp), %eax  # get first argument

    if:
       movl $1, %ebx
    cmpl %ebx, %eax
    jg if_greater
    movl %eax, %ebx
    imull %eax, %eax
    addl $1, %eax
    imull $2, %ebx
    subl %ebx, %eax
    jmp if_end

    if_greater:
       imull %eax, %eax
       subl $1, %eax

    if_end: leave
    ret

Exercise:
Write down the C code for the following example.

    /* solution */
    int expr(int x){
       if (x > 1)
          return (x*x - 1);
       else
          return (x*x + 1 - 2*x);
    }
while statement

- still compare and jump instructions

label:

... 

# while body 

cmp ..., ...

ej | jne |... label
Function that computes the linear combination of vectors a and b e.g.:

```assembly
.globl linear_comb
linear_comb:
    pushl %ebp              # save old base pointer
    movl %esp, %ebp         # set new base pointer
    movl 8(%ebp), %edx      # get dimension
    leal a, %ebx            # pa = &a
    leal b, %ecx            # pb = &b
    leal (%ebx,%edx,4),%esi
    movl $0, %eax
    while:
        movl (%ebx), %edi
        imull (%ecx), %edi
        addl %edi, %eax
        addl $4, %ebx       # pa ++
        addl $4, %ecx       # pb ++
        cmpl %esi, %ebx
        jl while
    leave
    ret
.data
    a: .long 3, 1, 2, 9, 5, 1, 4, 6, 5
    b: .long 1, 2, 7, 12, 5, 9, 4, 6, 3
```
Team Exercise

write an function in assembly that takes one argument \( x \) and computes the following function:

- \( f(x) = 25 \times x - 75, \) if \( x \geq 3 \)
- \( f(x) = 2 \times x + 1, \) if \( x < 3 \)