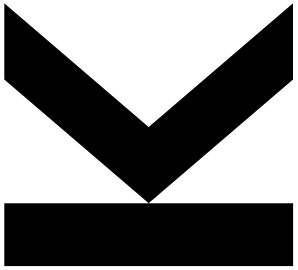
Exam Example Assembler



Systems Programming

JOHANNES KEPLER
UNIVERSITÄT LINZ

Michael Sonntag
Institute of Networks and Security

INSTITUTE
OF NETWORKS
AND SECURITY

Assembly example 1

- Write an x86 Assembler function that helps solving crossword puzzles: The first parameter is a string (zero-terminated) of a candidate word. The second parameter (also zero-terminated) is the “space” where this might fit in – with potentially a few letters already filled in. The function now verifies whether the first parameter matches the second:
  - They must be of exactly the same length. A length of zero, i.e. an empty string, is valid and two of these do match.
  - When the second parameter specifies a letter, this letter must appear at the same position in the first parameter.
  - If the second parameter shows a space (or whatever is defined as wildcard, see “PLACEHOLDER” below), any letter in the first parameter matches.
- Write an **assembly function** that checks this and returns 0 for a match and any other value if the two words no not match.

Assembly example 1

- The function's parameters and return type needs to conform to the following C function prototype:

```
int checkWord(char *candidate, char *slot);
```

- You **MUST**:
  - write the **implementation** of the function checkWord in assembly code (GNU syntax)
  - conform to the **C calling convention**

- You **need NOT**:
  - write **other parts** of the program, like the main program, or calling this function
  - perform any **error checking** on the parameters

Example In-/Output

- Examples (Spaces are shown as "□"):
  - **checkWord("Student","S□ud□□t") == 0**
  - **checkWord("Student","s□ud□□t") != 0**
  - **checkWord("Tree","S□ud□□t") != 0**
  - **checkWord("Students","S□ud□□t") != 0**
  - **checkWord("Student","S□ud□□t□") != 0**
- The following code sample shows the global constant that you should use in your implementation:

```
.equ  PLACEHOLDER,' '
      .section .text
      .type checkWord,@function
checkWord:
      # your code should start here
```

Assembly example 2

- Write an x86-64 Assembly function **sum(*n*)**, which **recursively** computes the sum of natural numbers from 1 to n:
- The function should return:
  - -1 if n < 0
  - 0 if n = 0
  - n + sum(n-1) in all other cases.
- The function's parameters and return type needs to conform to the following C function prototype:

```
int sum(int n);
```

Assembly example 2

- You **MUST**:
  - write the **implementation** of the function sum in assembly code (GNU syntax)
  - conform to the “SystemV AMD64 ABI” **calling convention** (i.e. the one used in the course), especially for the recursive calls
- You **need NOT**:
  - write **other parts** of the program, like the main program, or calling this function
  - perform any **error checks**, e.g. regarding overflows
- The following code sample shows the start of the function that you should use in your implementation:

```
        .section .text
        .type sum,@function
sum:
        # your code should start here
```

JOHANNES KEPLER
UNIVERSITÄT LINZ

INSTITUTE
OF NETWORKS
AND SECURITY

https://www.ins.jku.at

THANK YOU FOR YOUR ATTENTION!

Michael Sonntag
michael.sonntag@ins.jku.at
+43 (732) 2468 - 4137
S3 235 (Science park 3, 2nd floor)

JOHANNES KEPLER
UNIVERSITÄT LINZ
Altenberger Straße 69
4040 Linz, Österreich
www.jku.at