# WUT-4 Assembly Language

## Overview

This is the WUT-4 Assembly Language documentation.unlike most Assembly Language documents the op codes of the Assembly Language are not specified here. For the op codes see the WUT-4 architecture document.

The Assembly Language is line-oriented. Each line consists of an optional label, an opcode or directive, arguments (that may be optional), and an optional comment field. Arguments are separated from each other by spaces or optional commas. if commas are present, they are treated exactly like spaces.The optional comment field, if present, must be preceded by a semicolon. Labels, if present, must be followed by a colon.

Thus a typical line might appear:

label:  opcode argument argument, argument  ; some comment

Stylistically it is preferable to either use commas consistently or avoid their use. but the assembler does not enforce this.

As it assembles the file, the assembly maintains the value of dollar which is the next location to be filled in the output file. The assembler must actually maintain two values of dollar,One for the code segment and one for the data segment. Both values begin at 0. Values of dollar become virtual addresses in their respective segments.

The language supports constant expressions which may consist of numeric values, the operators
+, -, *, /, (truncating integer division), labels, and symbols defined set with the.set directive (see below).

Labels may be defined before they are used. but symbols created with `.set` must be defined before use.

## Output file format

The output file consists of a 16 byte header followed by the code segment words and then followed by the data segment (initialized data) if any. the header consists of a structure containing three 16-bit unsigned integers and 10 bytes (5 16-bit words) of reserved space. The first 16-bit unsigned integer of the file contains the value 0xDDD1. This is the “magic number” helpful for identifying WUT-4 executables. The second unsigned integer contains the number of file bytes that should be loaded in the code section. The third unsigned integer contains the

number of file bytes that should be loaded in the data segment, Data segment bytes immediately follow code segment bytes in the file.Although instructions in the code space must be aligned on 16-bit boundaries, no assumptions can be made about the size of the entire segment because it is perfectly legal to allocate static data in the code segment. So, for example, the code segment may end with string data having an odd length.

So if we imagine this pseudocode for the header:

```
struct header{
 magic: u16
 codesize:u16
 datasize:u16
 reserved: [5] u16
}
```

Then the file offset of the first byte of the code segment, if any, would be **`sizeof (header)`**. The file offset of the first byte of the data segment would be **`sizeof (header)+header.codesize`**.

# Directives

The assembly language supports the following directives.

| **Name** | **Arguments** | **Effect** |
|---|---|---|
| .align | one number. unit is bytes | Adjust the position of dollar for the current segment to a multiple of bytes given by the argument. The value may be any constant expression. |
| .bytes | list of byte values, or a quoted string.If a quoted string the data is taken as utf-8 encoded. | Place the bytes successively in the current segment at dollar, incrementing dollar. Each 8-bit value may be any constant expression. Values which do not fit in 8 Bits are truncated with a warning. |
| .words | list of 16-bit word values | Each 16-bit value may be any constant expression. |
| .space | one number. Unit is bytes. | Iincrement dollar by the argument, which may be any |

| | | |
|---|---|---|
| | | constant expression. |
| .code | none | Make the code segment the current segment. |
| .data | none | Make the data segment the current segment |
| .set | symbol name , value | Define the symbol to represent the value.All such Define symbols are constants and may not be reset.The value may be any constant expression. |
| | | |
| | | |
| | | |