WAVES ASSEMBLY

LANGUAGE MANUAL

First Edition

Israel Alagbe

Waves Technology

Introduction

All microprocessors have a set of features that programmers use. In most instances, a programmer will not need an understanding of how the processor is actually constructed, meaning that the wires, transistors, and/or logic boards that were used to build the machine are not typically known. Each family of processors has its own set of instructions for handling various operations like getting input from keyboard, displaying information on screen called machine language instruction which processor understands. However machine language is too complex for using in software development. Low level assembly language is designed for a specific family of processors that represents various instructions in symbolic code and a more understandable form.

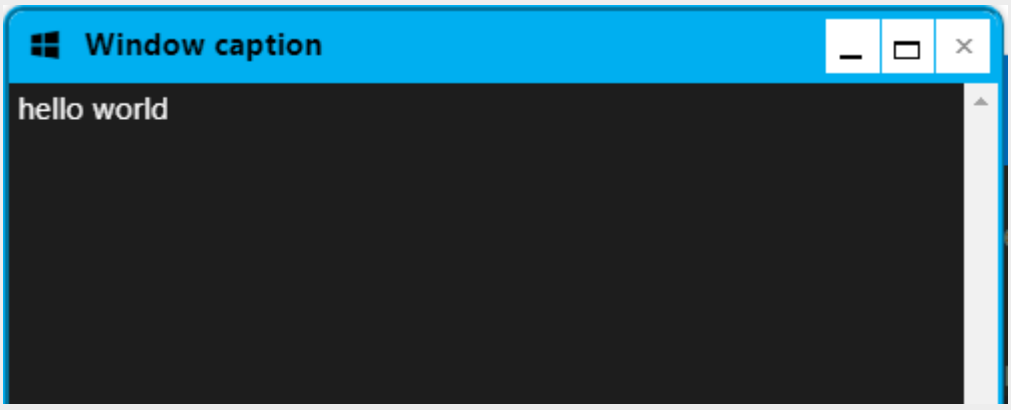
Learning of assembly languages for students and beginner programmers has been very difficult due to their difficult syntax and because they were not designed for beginner programmers. Most assembly languages also assumes that programmers knows what they are doing. Waves assembly has been designed to tackle this problem by providing a controlled environment for learning assembly languages.

Waves Assembly basic syntax

A hello world program example:

```
1  .data
2      .string str "hello world" ;Declare the string constant
3  .end
4  main:
5      get r0 str ;Get the address of the string
6      call printStr r0 ;Call the printString function
```

Program output:



Data section

The data section is used for declaring constants. This data does not change at runtime. You can declare various constant values in this section

```
.data
    ;Declare your data here
.end
```

Procedure section

The procedure section is used for keeping the actual code. This section must contain a main procedure which tells the virtual machine where the program execution begins.

```
1 main:
2     ;Your code goes here
```

Comments

Waves assembly language comment begins with a semicolon (;). It may contain any printable character including blank.

It can appear on a line by itself, like:

```
1 ;This is ignore by the compiler
```

or, on the same line along with an instruction, like:

```
1  add r0 r1 r2 ; add r0 and r1 and put the result in r2
```

Format of Waves assembly language

```
[label:] [.local label:] mnemonic [operands] [;comment]
```

Waves assembly registers

Register is a computer memory that is used to quickly accept, store, and transfer data and instructions that are being used immediately by the CPU. It actually makes sense to have registers as it makes sense to have office desks so that you can easily identify tasks you are currently working on. Register enables computer to have access to memory more easily than accessing the RAM.

Waves assembly has 65535 registers in which 0-254 registers are actually accessible at a time by the user. The other registers are used when a stack frame is being created. A new stack frame is created when a function is being called and it allocates 254 registers for the function and it is available throughout the lifetime of the stack frame.

There are no special register in waves assembly language as all registers are created equal. The register are prefixed with letter ‘r’ e.g r1, r2, r234. Waves register can only contain a 32bit integer number.

Simple register arithmetic:

```
1  main:
2      iconst r0 100 ;Load 100 into register 0
3      iconst r1 50  ;Load 50 into register 1
4      add r0 r1 r2  ;r2=r1+r2
5      call printInt r2 ;print integer content of register 2
```

Moving value from one register to another:

```
2      move r0, r1 ;Move the value in register 0 to register 1
```

Waves assembly built-in functions

Built in functions in waves assembly language are written to abstract the low level functionality of waves system function call. It has an high level syntax that is very easy to understand.

A function can be called like so:

```
1 ;Call printInteger function with register 0 as argument
2 call printInt r0
```

Or

```
1 ;Call alloc function with no argument
2 call alloc
3 ;call average function with 3 arguments
4 ;[r0,r1,r2]
5 call average r0-r2
```

Built-in functions and signatures:

printInt(int value):void ;print an integer value in register

printChar(int char):void ;print an character value in register

printStr(int address):void ;print a string value from the address

alloc(int size):int address ;Allocate an amount of memory and return its address