Wallcomp Assembly Specification

Samuel Russell

October 21, 2016

It is designed to

- be very close to the machine code, with the assembler only really handling the layout of instructions and data in memory.
- to make it simpler to write the assembler that isn't that flexible in terms of syntax.

1 Labels

Labels go on their own line and label the next line which is an instruction.
They are written with...

1. no spaces before,
2. then an 'L',
3. then an identifier not containing spaces,
4. then a new line

You can define more than one label on consecutive lines and they will be given the same value.
Eg.

```
Linit
Lloop
```

2 Instructions

All instructions like the machine's instructions have 1 operand. Instructions are written with...

1. at least one space at the start of the line,
2. then the acronym,
3. then at least one more space,
4. then the operand,
5. then a new line (with no trailing spaces).

Eg.

```
 BR 1
```

or

```
 OPR ADD
```

The instructions are as follows taken from the machine code specification.
Access to constants and program addresses is provided by instructions which either load values directly or enable them to be loaded from a location in the program:

```
LDAM: areg <- mem[oreg] load from memory
LDBM: breg <- mem[oreg] load from memory
STAM: mem[oreg] <- areg store to memory

LDAC: areg <- oreg load constant
LDBC: breg <- oreg load constant
LDAP: areg <- pc + oreg load address in program
```

Access to data structures is provided by instructions which combine an address with an offset:

```
LDAI: areg <- mem[areg + oreg] load from memory
LDBI: breg <- mem[breg + oreg] load from memory
STAI: mem[breg + oreg] <- areg store to memory
```

Branching, jumping and calling The branch instructions include conditional and unconditional relative branches. A branch using an offset in the stack is provided to support jump tables.

```
BR: pc <- pc + oreg branch relative unconditional
BRZ: if areg = 0 then pc <- pc + oreg branch relative zero
BRN: if areg < 0 then pc <- pc + oreg branch relative negative
BRB: pc <- breg branch absolute
```

Inter-register operandless Instruction: OPR

```
OPR ADD: areg <- areg + breg add the registers values
OPR SUB: areg <- areg - breg subtract the registers values
```

Operands

Operands can be written as either:

- decimal format with negative value being prefixed by a '-' sign.
- unsigned hexadecimal format prefixed by '0x' to indicate this.
- the ORP command's sub instruction can be written by using the mnemonic ADD and SUB.

Data Sections

Slightly differently, to define a data word (16bits) to be stored to memory. we use

```
DATA {initial value of the 16bit word}
```

It is suggested you label this word with a label and refer to it using that since the assembler might move it (for example to align it to fit in 1 16 bit word).

Using labels as Operands for Instructions

There are some instructions which use labels as operands.

```
LDAM, LDBM, STAM, LDAC, LDBC, BR, BRZ, BRN, LDAP
```

Instead of writing a static number you can use a label by typing 'L' and then the identifier.

The first 5 follow the previous definitions with labels, But BR, BRZ, BRN, LDAP do a little extra, with as you would expect, the assembler calculating the relative jump to the specified label and using that in the machine code.

3 Comments

Any line that starts with '-' will be ignored so can be used for comments.
Eg.

```
-this is a comment
```

4 Example Program

```
-This program maintains a stack while calling a function that returns 5
 BR       L3
Lsp
 DATA     0x7CFE
L3
 LDAP     L5
 BR       L4
L5
 LDAI     1
 BR       -2
L4
L2
 LDBM     Lsp
 STAI     0
 LDAC     -1
 OPR      ADD
 STAM     Lsp
 LDAC     5
 LDBM     Lsp
 STAI     2
 LDAC     1
 OPR      ADD
 STAM     Lsp
 LDBI     1
 BRB      0
```

5 Notes

- The assembler doesn't like trailing white-space.
- BR -1 probably won't do what you think since the negative value needs an extra instruction to produce it. (So should use -2)