# The BigHex Machine Assembly Examples

#### Samuel Russell

November 9, 2016

### 1 Adding Two Constants

First you must load the two constants in to the two registers (areg and breg) and then perform the interegister add operation.

```
LDAC 1
LDBC 2
OPR ADD
```

The first line loads the number one in to areg, the second line loads two in to breg and the third adds together the value of areg and breg and puts the answer in to areg.

Note that each instruction goes on a new line and has at least one space at the beginning.

## 2 Adding Two Values from Memory

A data portion is allocated in memory and the address of it (using a label) is used to load the value in to the registers before the add operation (used as before) and store back to memory afterwards.

```
LDAM Lval_one
LDBM Lval_two
OPR ADD
STAM Lval_out

Lval_one
DATA 0x10
Lval_two
DATA 2
Lval_out
DATA 0
```

Note we can also use hexadecimal representation to define the values.

## 3 Summing the Integers from 1 to N

For this operation we will create a loop using the conditional and unconditional branch instructions. We will need to keep track of the running total and a variable to count the number of iterations. Since we only have two registers we must store these values in our main memory and use loads and stores as we manipulated them. We use labels to mark lines of code to jump to.

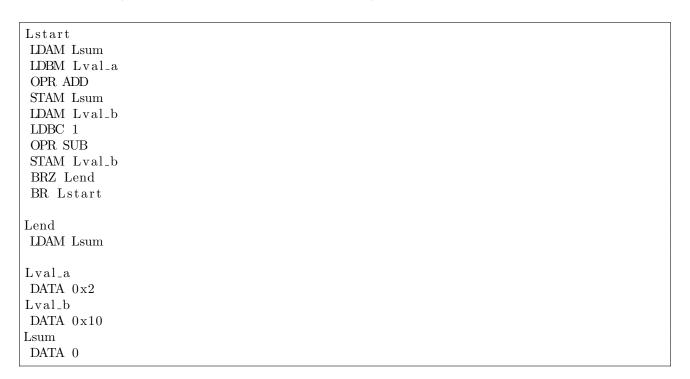


This code can be made shorter by utilising the fact that the loop counter is already in breg when we need to decrement it.

Lstart LDAM Lsum LDBM Lcounter OPR ADD STAM Lsum LDAC -1OPR ADD STAM Lcounter BRZ Lend BR Lstart Lend LDAM Lsum Lcounter DATA 10 Lsum DATA 0

# 4 Multiplication

This machine has a very RISC architecture which means instructions can be decoded simply and the control logic is relatively small. This does mean however that there is no multiplication instruction and so it has to be done iteratively (this was true up until surprisingly recently).



#### 5 Factorial

We can then use this multiplication code we have written to write a program that calculates the factorial of a given number. (remember n! = (n) \* (n-1)...(2) \* (1))

```
-jump straight to code
BR Lstart
-Allocate space to store variables
Lval_a
DATA 0x0
Lval_b
DATA 0x0
Lmult_sum
DATA 0x0
Lcount
DATA 4
Ltotal
DATA 1
-this bit loads the parameters for our multiplier from our total variables
Lstart
LDAM Ltotal
STAM Lval_a
LDAM Lcount
STAM Lval_b
LDAC 0
STAM Lmult_sum
- this is the call to multiplier
BR Lmult_start
-this bit receives the code from multiplier,
-then checks if we need to go round again.
Lcheck_finish
LDAM Lmult_sum
STAM Ltotal
LDAM Lcount
LDBC 1
OPR SUB
STAM Lcount
BRZ Lend
BR Lstart
-holds the processor in an infinite loop
Lend
LDAM Ltotal
BR Lend
-this is the multiplier code as before
Lmult_start
LDAM Lmult_sum
LDBM Lval_a
OPR ADD
STAM Lmult_sum
LDAM Lval_b
LDBC 1
OPR SUB
STAM Lval_b
BRZ Lmult_end
BR Lmult_start
-now we return to the previous code
Lmult_end
BR Lcheck_finish
```

This piece of code calculates factorial 4. It works well but some improvements could be made to make it run faster.