Explanation of Sandbox

and proof that it is Turing complete

Patrick Lin

September 17, 2013

We use the following terminology:

- [X] means the value at memory location X.
- $X denotes that a memory address is desired.
- #L denotes that a label is desired. If a label is used in a Branching instruction it must be set somewhere using a Label instruction.
- N denotes an integer is desired.
- Memory addresses range from 1 to 1048576. If the address is negative, then the value stored at the positive value of the address is used instead. For example: if [20]=3, then 11COP -20 10" will be translated to 11COP 3 10"

The Sandbox, as of 17 Sep 2013, supports the following functions in its program files:

- ADD $t $s1 $s2 // [$t]←[$s1]+[$s2]
- SUB $t $s1 $s2 // [$t]←[$s1]-[$s2]
- COP $t $s // [$t]←[$s]
- SET $t N // [$t]←N
- BEQ #L $s1 $s2 // IF [$s1]=[$s2] GOTO #L
- Equivalents for BNE ($\neq$), BG ($>$), BL ($<$), BGE ($\geq$), BLE ($\leq$)
- DISP $t // Displays [$t]
- LABEL #L // Create Label #L
- EXIT N // Exits if N=0, otherwise prints "Return code N" first

This is a simple mode lthat losely follows assembly syntax.

It also (theoretically) supports preloading items into memory, but this has not yet been tested.

The Sandbox would Turing Complete if we assumed an infinite memory, instead of being limited to 1048576 locations.

Proof. A Turing machine needs to support the following operations:

- Read the value at the current location on the tape
- Based on the read value and the current state:
  - Write a value to the current location to the tape (can be the same as before)
  - Change to a new state (can be the same as before)
  - Move left or right on the tape (some versions allow non-movement)

All such programs can be simulated on this sandbox using a variety of SETs, ADDs, SUBs, and BEQs:

First, we use two fixed locations to hold the current location and the current state.

We need to see if the pair of the read value and the current state corresponds to a rule. To do this, for each rule, we can SET the value and state to some unused place in memory, and then see if the read value and current state match using BEQs. Then, we can use SETs to set the new value and state, and use ADD/SUB to move the location right/left.

Thus, any Turing machine can be simulated on the sandbox. □