Appendix x.1

Henrik Sommerland, Oskar Ahlberg, Aleksander Lunqvist

March 5, 2014

Abstract

In this file we will go thru the functions of the Components.sml, and explain the functions that we are using.

Contents

1 Introduction

The following structures and signatures are present in Components are the Ram, Stack, Register and ProgramCounter. The structure can be seen as a new part of the library in SML, we will now go thru the structures and the encapsulated functions.

2 Components.sml

2.1 The Ram structure

2.1.1 Synopsis

```
signature RAM
structure Ram :>RAM
```

The Ram structure provides a base of the functions of a ram memory that is a interglacial part of the virtual machine.

2.1.2 INTERFACE

```
type memory = int array
val initialize :  int → memory
val getSize :  (memory) → int
val write :(memory * int * int ) → memory
val read :  (memory * int) → int
val load :  (memory* int list) → memory
val writeChunk :  (memory * int * (int array)) → memory
val readChunk :  (memory * int * int) → int array
val dump :  memory → string
```

2.1.3 Description

```
val initialize :  int → memory
```

Initialize the ram to a memory with the size of int, when int $>$ 0

```
val getSize :  (memory) → int
```

Gets the size of the ram (memory)

```
val write :(memory * int * int ) → memory
```

write takes a memory and writs a new value of int at the pointer of the first int and returns the memory

```
val read:  (memory * int) → memory
```

read takes a memory and reads the value of the place of int

```
val load:  (memory * int list) → memory
```

load takes a list of values and loads them to the memory

```
val writeChunk:  (memory* int *( int array)) → memory
```

writeChuck takes a memory and a start pointer and adds a chunk to the memory

`val readChunk: (memory * int *int) → int array`

readChumk takes a memory and reads a chunk form first int to the last int and gives the values as an int array

`val dump: memory → string`

dump takes a memory and returns the value as strings

This concludes the RAM structure

2.2 The Stack structure

2.2.1 Synopsis

```
signature STACK
structure Stack :>STACK
```

The Stack structure provides a base for the stack part of the Pc structure.

2.2.2 INTERFACE

```
datatype stack = Stack of (int list)
val empty : stack
val push : stack * int → stack
val pop : stack → stack
val top : stack → int
val isEmpty : stack → bool
val dumpStack : stack → string
```

2.2.3 Description

`val empty : stack`

is a definition of a empty Stack

`val push : stack * int → stack`

takes a stack and adds the value of int to the stack.

`val pop : stack → stack`

takes a Stack and "pop"s the first element of the stack.

`val top : stack → int`

takes the stack and returns the first element of the stack

`val isEmpty : stack → bool`

takes a stack and checks if it is empty if it is then true else false.

`val dumpStack : stack → string`

takes a stack, then pops the stack until its empty and returns all values as string

This concludes the stack structure.

2.3 The Register structure

2.3.1 Synopsis

```
signature REGISTER
structure Register :>REGISTER
```

The Register structure provides a base structure of the different register that is contained in the Pc as well the Virtual machine. The vm has two different registers.

2.3.2 INTERFACE

```
datatype reg = Reg of int
val setData : (reg * int) → reg
val getData : reg → int
val increment : reg → reg
val decrement : reg → reg
val dumpRegister : reg → string
```

2.3.3 Description

```
val setData : (reg * int) → reg
```
Setups a new Register
```
val getData : reg → int
```
Gets the value of the reg as an int
```
val increment : reg → reg
```
Takes a reg and increment it with one.
```
val decrement : reg → reg
```
Takes a reg and decrements it with one.
```
val dumpRegister : reg → string
```
Takes the register and adds all elements to a string.

This concludes the Register structure.

2.4 The Program_Counter structure

2.4.1 Synopsis

```
signature PROGRAM_COUNTER
structure ProgramCounter :>PROGRAM_COUNTER
```

The ProgramCounter structure provides the foundation of the Virtual machine this is the hearth of the

2.4.2 INTERFACE

```
datatype pc = Pc of (int * Stack.stack * Register.reg * Register.reg)
val incrementPointer : (pc * int) → pc
val jump : (pc * int) → pc
val subroutineJump : (pc * int) → pc
val return : pc → pc
val interrupt : (pc * int) → pc
val dumpPc : pc → string
```

2.4.3 Description

`val incrementPointer : (pc * int) → pc`
Takes a Pc and adds a int > 0
`val jump : (pc * int) → pc`
Takes a Pc and jumps the pc counter to the value of int > 0
`val subroutineJump : (pc * int) → pc`
Takes a Pc and preforms SubrutineJump with the value of int > 0 and adds the value of the pointer $+ 1$ to the stack
`val return : pc → pc`
Takes a pc and gets the value from the pointer and pops the stack with the value
`val interrupt : (pc * int) → pc`
if the value of a is 1 or 2, then the value of i is added to s
`val dumpPc : pc → string`
Takes a pc and dumps the content of the pc as a string (the Pc contained a pointer, Stack, and tow registers)

This concludes the ProgramCounter structure