An Assembler for AVM2 using S-Expression

Takashi Yamamiya

This material is based upon work supported in part by the National Science Foundation under Grant No. 0639876. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

VPRI Memo M-2009-010

Viewpoints Research Institute, 1209 Grand Central Avenue, Glendale, CA 91201 t: (818) 332-3001 f: (818) 244-9761

An Assembler for AVM2 using S-Expression

Takashi Yamamiya

takashi@vpri.org

2009-10-04

Overview

ABCSX is an assembler and disassembler for the ActionScript Virtual Machine 2 (AVM2) [1] and the ActionScript Byte Code (ABC). It runs on Cola/Amino language or PLT-Scheme. The syntax consists of s-expressions and a program can be constructed with normal list operations in Scheme like language. The goal of this utility is to build a high level language compiler for Adobe Flash Player. To get the idea, "Hello World!" programs for both ABCSX and abcasm (a standard assembler utility consisted in the AVM2 source tree [4]) are shown.

```
;;;; A "Hello World!" program in ABCSX ASM-form
(asm
 (method
  (((signature
     ((return_type *) (param_type ()) (name "hello")
      (flags 0) (options ()) (param_names ())))
    (code
     ((getlocal 0)
      (pushscope)
      (findpropstrict ((package "") "print"))
      (pushstring "Hello, World!!")
      (callproperty ((package "") "print") 1)
      (returnvoid))))))
 (script (((init (method 0)) (trait ())))))
```

```
// A "Hello world World!" program in abcasm
function hello():*
{
    getlocal 0
    pushscope
    findpropstrict print
    pushstring "Hello, World!!"
    callproperty print (1)
    returnvoid
}
```

Although a program written in abcasm syntax is more concise than ABCSX, the semantics is rather ambiguous. For example, in spite of each symbol name in ABC belongs to namespace(s), the syntax of abcasm doesn't describe it clearly. In this case, "print" is implicitly interpreted to a Multiple Namespace Name with a namespace set including PackageNamespace with no name. In case of

ABCSX, it is explicitly represented as PackageNamespace with no name by `((package "") "print")`. This implicit behavior might be useful for writing a program by hand, but not necessary for a machine generated code. ABCSX rather takes a direction toward verbose but unambiguous style.

ABCSX offers two forms of syntax. ASM-form is higher level syntax introduced above. ABC-form is identical to an abstract syntax tree of ABC binary file. This is useful when exact behavior is need to know while debug.

```
;;;; A "Hello World!" program in ABCSX ABC-form
(abc
 (minor_version 16)
 (major_version 46)
 (constant_pool
  ((integer ())
   (uinteger ())
   (double ())
   (string ("hello" "" "print" "Hello, World!!"))
   (namespace ((package (string 2))))
   (ns_set ())
   (multiname (((namespace 1) (string 3))))))
 (method (((return_type (multiname 0)) (param_type ())
           (name (string 1)) (flags 0) (options ()) (param_names ()))))
 (metadata ())
 (instance ())
 (class ())
 (script (((init (method 0)) (trait ()))))
 (method_body
  (((method 0) (max_stack 2) (local_count 1)
               (init_scope_depth 0) (max_scope_depth 1)
    (code
     ((getlocal 0)
      (pushscope)
      (findpropstrict (multiname 1))
      (pushstring (string 4))
      (callproperty (multiname 1) 1)
      (returnvoid)))
    (exception ())
    (trait ())))))
```

Using ASM-form, a compiler writer doesn't have to care about building a constant pool, or code hint information (AVM2 requires a frame information like stack size and register size used in a code).

Background

One of goals of the STEPS project [3] and COLA programming language is to provide full control of computer environment from application level to machine language level, so that users could experiment and design their own programming language best fit to their task. It also will be used as a basis of next generation of EToys programming environment for kids.

We chose Adobe Flash Player as one of platforms of the system because of its popularity and usability. Using Flash's virtual machine on a web browser, we could deliver our programming

environment without concerning about installation or security issue.

AVM2 has some disadvantages compared to Java VM. AVM2 lacks multi task support, and its dynamic dispatching function is relatively slow. But the startup speed and memory footage are good, and these aspects are essential to casual users. Especially AVM2 will be good platform to implement EToys.

ABCSX is designed to be a back end module for COLA, command line assembler / disassembler, and a Scheme library. While it is a part of COLA/ABC compiler, it also can be used as a command line tool to examine and debug ABC binary file.

Usage

Command line tool

A version of ABCSX is publicly available on the github repository [2]. It includes command line tools run on PLT-Scheme. There are also example programs at `examples/` directory. The assembler and disassembler use same file format and the assembler `asm.ss` can read an output file generated by disassembler `dump.ss`.

`asm.ss` *`filename.sx`*

Generate an ABC binary file from ASM-form or ABC-form. The output file name is filename.sx.abc.

`dump.ss [-abc]` *`filename.abc`*

Disassemble an ABC binary file. The output is printed to stdout. If `-abc` option is specified, ABC-form is chosen as output format.

`runasm.sh` *`filename.sx`*

Assemble ASM-form or ABC-form and execute it by avmshell. It requires avmshell installed. Avmshell is included in Tamarin VM's source tree [4].

`swf_abc.erl` *`width height classname abcfile.abc`*

A helper program to generate a flash file from an abc file. It requires Erlang.

Function

`(write-asm` *`list port`*`)` procedure

Assemble ASM- or ABC-form to a binary stream.

`(read-asm` *`port`*`)` procedure

Disassemble a binary stream to ASM-form.

`(from-asm` *`list`*`)` procedure

Convert ASM-form to ABC-form. This is a part of process of assemble. Each literal value is replaced to a reference, and a constant pool is created

`(to-asm list)` procedure

Convert ABC-form to ASM-form. This is a part of process of disassemble. Each constant reference in the ABC-form is replaced to a literal value based on the constant pool.

Data Type

ABC's data is expressed as scheme expression in ABCSX. In ASM-form, data conversion has subtle context dependency in code-subsection.

- integer - An integer value in Scheme is converted to ABC integer value depend on the context.
  - int (s32) - In code-subsection, an integer is converted to a signed 32 bit integer if the opcode requires integer e.g. `pushint`.
  - uint (u32) - In code-subsection, an integer is converted to a unsigned 32 bit integer if the opcode requires integer e.g. `pushuint`.
  - u30 - An integer is converted to a unsigned 30 bit integer in ABC anywhere else.
- double (d64) - A floating point number value is converted to a 64-bit double precision IEEE 754 value.
- string - A string is converted a string value in ABC.
- namespace - Some list expressions are converted to namespace values in ABC. The format is (*kind string*). For example, `(package "org.vpri")` is converted to a package namespace named `"org.vpri"`.
  - Namespace - `(ns string)` is converted to Namespace
  - PackageNamespace - `(package string)` is converted to PackageNamespace
  - PackageInternalNs - `(internal string)` is converted to PackageInternalNs
  - ProtectedNamespace - `(protected string)` is converted to ProtectedNamespace
  - ExplicitNamespace - `(explicit string)` is converted to ExplicitNamespace
  - StaticProtectedNs - `(static string)` is converted to StaticProtectedNs
  - PrivateNs - `(private string)` is converted to PrivateNs
- namespace set - A namespace set can not be described as a literal. Instead, it is declared in a constant pool of ns_set-section at first, and be made reference by index e.g. `(ns_set 1)`.
- multiname - Some list expressions are converted to multiname (symbol) in ABC.
  - QName - `(namespace string)` is converted as QName e.g. `((package "flash.display") "Sprite"))`
  - RTQName - is not supported.
  - RTQNameL - is not supported.
  - Multiname - `((ns_set integer) string)` is converted as a Multiname e.g. `((ns_set 1) "addChild")`
  - MultinameL - is not supported.

Syntax

The syntax of ASM-form is explained. ABCSX uses same symbol names as "ActionScript Virtual Machine 2 (AVM2) Overview" unless it is too strange. Especially, underline delimited names and capital names are derived from the document.

ASM-form

`(asm` *`[ns_set-section] method-section [metadata-section] [instance-section] [class-section] script-section)`*

ASM-form begins with a symbol `asm`, and contents are followed. ns_set-section, instance-section, and class-section are optional.

ns_set-section

`(ns_set (ns_set` *`namespace ...`*`) ...)`

Ns_set-section will be a part of constant pool, and it is only necessary if namespace set is used in other part of the ASM-form. You can not specify a namespace set directly as a literal, but you need to define it in ns_set-section and point it with the index number.

Ns_set-section begins with a symbol `ns_set` and a list of ns_set_info is followed. A ns_set_info begins with a symbol `ns_set` and it includes a list of namespaces. A namespace set is referred with one-based index by other part. For example, the first namespace set is referred as `(ns_set 1)`.

method-section

`(method (`*`signature-subsection code-subsection`*`) ...)`

Method-section includes a list of pairs of signature and code. A method is referred by zero-based index. For example, the first method is referred as `(method 0)`.

signature-subsection

`(signature (return_type` *`multiname`*`) (param_type (`*`multiname ...`*`)) (name` *`string`*`) (flags` *`integer`*`) (options (`*`option...`*`)) (param_names (`*`multiname ...`*`)))`

Signature-subsection describes method's signature. If * is specified at the return_type. It is treated as Any Type. A name entry is not used as a method name in a program. In a typical case, methods are explicitly bound to named slots in initialization code at script-section or object constructor.

code-subsection

`(code (`*`instructions...`*`))`

Code subsection describes a sequence of instruction code of the method. A label is specified as a symbol, and normal instruction is specified as a list as:

`([`*`offset-number`*`]` *`inst-name args ...`*`)`

offset-number is optional and used just as a place holder. It can be a integer or symbol _. ABCSX's

disassembler put a byte offset number at this place, but the assembler ignores it.

metadata-section

```
(metadata (metadata_info ...))
```

Metadata-section describes a list of metadata entries.

instance-section

```
(instance (((name multiname) (super_name multiname) (flags integer) (interface
(multiname ...)) (iinit method) (trait (trait_info ...)) ...)))
```

Instance-section describes a list of class definitions. Class members are defined by a list of trait_info.

class-section

```
(class (((cinit method) (trait (trait_info...))) ...))
```

Class-section describes a list of static members of class definition. The number of this list is same as instance-section, and each entry of class-section corresponds to instance-section. A definition consists of a class initializer and trait_info definitions.

script-section

```
(script (((init method) (trait (trait_info...))) ...))
```

Script-section defines a list of static functions. It is also used as a program's startup code. Once the virtual machine reads a program, the last entry of script-section is invoked. Each entry consists of a method reference and a list of trait_info. Trait_info is used as a function's environment.

trait_info

Trait_info defines a fixed property of an object, class, or method. ABCSX only supports Trait_Slot and Trait_Class.

Trait_Slot

```
((kind slot) (name multiname) (slot_id integer) (type_name multiname) (vindex
integer) (vkind integer) (metadata (metadata_info...)))
```

Trait_Slot defines a named slot in the context.

Trait_Class

```
((kind class) (name multiname) (slot_id integer) (classi class) (metadata
```

`(`*`metadata_info`*`...)))`

Trait_Class defines a named slot with a class in the context.

metadata_info

`((name` *`string`*`) (items (((key` *`string`*`) (value` *`string`*`)) ...)))`

Metadata_info defines an entry including arbitrary key/value pairs.

Current Status

Currently, only major elements in AVM2 are implemented.

- All primitive data types are implemented.
- 75 instructions (about a half of the whole instruction set) are implemented.
- Only QName (Qualified Name) and Multiname (Multiple Namespace Name) are implemented.
- Optional parameters or parameter names are not implemented.
- Trait_Method, Trait_Getter, Trait_Setter, Trait_Function, or Trait_Const are not implemented.
- Exception is not implemented.

Example

As a complete example, A GUI version of "Hello World!" program is shown with commentary. This file is available at `examples/textField.sx` on the source tree.

```
(asm
 (ns_set
  ((ns_set (package "") (package "flash.text"))))
```

An ASM-form begins with a symbol `asm`, and a ns_set-section follows if necessary. This example declare one namespace set including package namespaces `""` and `"flash.text"` as `(ns_set 1)`. Ns_set's index number starts with 1 because this is a member of constant pool. Other kind of index number (method, class) starts with 0.

```
 (method
  (((signature ((return_type *) (param_type ()) (name ""))
               (flags 0) (options ()) (param_names ())))
    (code
     ((returnvoid))))
```

The first method is referred as `(method 0)`. It is used as a class initializer in the class-section, but nothing to do in this case.

```
   ((signature ((return_type *) (param_type ()) (name "")
```

```
              (flags 0) (options ()) (param_names ())))
   (code
    ((getlocal_0)
     (pushscope)
     (getlocal_0)
     (constructsuper 0)
     (findpropstrict ((ns_set 1) "TextField"))
     (constructprop ((package "flash.text") "TextField") 0)
     (coerce ((package "flash.text") "TextField"))
     (setlocal_1)
     (getlocal_1)
     (pushstring "Hello, World!")
     (setproperty ((package "") "text"))
     (findpropstrict ((package "") "addChild"))
     (getlocal_1)
     (callproperty ((package "") "addChild") 1)
     (pop)
     (returnvoid))))
```

The second method is later used in the instance-section as class Hello's constructor. It builds an instance of `flash.text.TextField` and set "Hello, World!" to the property named `text`. Finally, the text field is added to this (Hello) object.

```
  ((signature ((return_type *) (param_type ()) (name ""))
              (flags 0) (options ()) (param_names ())))
   (code
    ((getlocal_0)
     (pushscope)
     (getscopeobject 0)
     (findpropstrict ((package "") "Object"))
     (getproperty ((package "") "Object"))
     (pushscope)
     (findpropstrict ((package "flash.display") "Sprite"))
     (getproperty ((package "flash.display") "Sprite"))
     (pushscope)
     (findpropstrict ((package "flash.display") "Sprite"))
     (getproperty ((package "flash.display") "Sprite"))
     (newclass 0)
     (popscope)
     (popscope)
     (initproperty ((package "") "Hello"))
     (returnvoid))))))
```

The third method is used as the startup script. It creates an environment and initialize a new class defined in instance-section and class-section by `newclass` instruction.

```
(instance
 (((name ((package "") "Hello"))
   (super_name ((package "flash.display") "Sprite"))
   (flags 0)
   (interface ())
   (iinit (method 1))
   (trait ()))))
```

```
(class (((cinit (method 0)) (trait ()))))
```

Instance-section and class section define classes. In this case, A class named `Hello` is defined as a subclass of `flash.display.Sprite`. When a SWF file is created from ABC file, a SymbolClass tag in the SWF creates association between a class name defined here and the main timeline of the SWF. In ABCSX tool set, script `swf_abc.erl`'s third argument does this task.

```
(script
 (((init (method 2))
   (trait
    (((kind class)
      (name ((package "") "Hello"))
      (slot_id 1)
      (classi (class 0))
      (metadata ()))))))))
```

Script-section defines the startup script and predefined named slot.

References

- [1] ActionScript Virtual Machine 2 (AVM2) Overview. http://www.adobe.com/devnet/actionscript/articles/avm2overview.pdf
- [2] ABCSX github repository. http://github.com/propella/abcsx
- [3] Steps Toward the Reinvention of Programming (First Year Progress Report). http://www.vpri.org/pdf/tr2007008_steps.pdf
- [4] Tamarin Project http://www.mozilla.org/projects/tamarin/