Concordia University
Department of Computer Science and Software Engineering

SOEN422 Fall 2021 Project [40%]

—

Porting a Small Edison Virtual Machine to the Arduino Nano

Purpose

This project aims to allow students an opportunity to practice a port of a virtual machine (VM) to small-footprint embedded systems: the Arduino Nano. The VM's instruction set is tailored to support a subset (operators, basic statements var/if/while, and functions) of a subset of the Edison programming language, intended for a restrictive microcontroller environment ATmega368P 8-bit microcontroller with 32K bytes Flash and 2K bytes SRAM. This project will require six (6) tasks to incrementally achieve the port on target by using the following roadmap:

- Task 1 [%3]: Complete the AUnit tool on Arduino Nano
- Task 2 [%3]: Do a small shell to dump memory on host and target
- Task 3 [%8]: Do a first VM port on your host development platform
- Demo 1 [%4]: Demo tasks 1-2-3, VM on host, and code review (**Wednesday, November 17**)
- Task 4 [%4]: Do a second VM port on the target
- Task 5 [%9]: Replace the host loader with a target serial loader
- Demo 2 [%4]: Demo tasks 4-5, VM on target, and code review (**Wednesday, November 24**)
- Task 6 [%5]: Do your project report

Before Getting Started: Important Basic Rules to Follow

Before tackling the tasks, your team must respect some important basic rules to fulfill the VM port requirements. I will explain (during lecture time) the details of the requirements for all tasks. **You should use both AUnit testing tools to regress all your tests from day one up to the due dates (no other testing tools will be accepted during your demos and final release)**:

- AUnit tool for your host platform (already given at the beginning of the semester)
- AUnit tool for Arduino Nano (to be completed in Task #1, see the source file `TestAUnit_todo.c`)

Regarding `Task 5`, **you must use the Lecture 8 `SerialLoader.cs` in C# on the host side. It has been provided to focus on the task to be done on the target side.** No other loaders, tools, or languages (such as C++, Java, Python) will be permitted on the host side.

The regression of the tests is a crucial element of success for the port of the VM. During the progress of the project's tasks, your team must continuously ensure that the VM's port on your platform is identical with that of the target, except for the HAL and BSL layers, which are hardware and platform dependent. In other words, 90-95% of the C code must be self-tested continuously to ensure that they work well on both platforms.

A Java binary version of the Small Edison compiler is provided, and a suite of pre-compiled Small Edison (15) programs for your team to port. The main reason for not providing the compiler's source code is to ensure that every team focuses on the virtual machine code and its serial memory loader. At the end of the semester, students interested to help on adding features to the language subset can contact me directly.

Here is an example of my batch file `_runAllPic.bat` that exercises the Small Edison VM by running each test of the suite (from `f0.pic`[1] to `r0.pic`) concatenating their corresponding output to a text file `_allPicTests.txt`," which will be verified by the `aunit` testing tool program on host.

```
@echo Running all Small Edison (15) programs on Host.
del -f _allPicTests.txt

se f0.pic >  _allPicTests.txt
se f1.pic >> _allPicTests.txt
se f2.pic >> _allPicTests.txt
se f3.pic >> _allPicTests.txt
se f4.pic >> _allPicTests.txt
se f5.pic >> _allPicTests.txt
se f6.pic >> _allPicTests.txt
se a0.pic >> _allPicTests.txt
se a1.pic >> _allPicTests.txt
se a2.pic >> _allPicTests.txt
se a3.pic >> _allPicTests.txt
se c0.pic >> _allPicTests.txt
se c1.pic >> _allPicTests.txt
se m0.pic >> _allPicTests.txt
se r0.pic >> _allPicTests.txt
type         _allPicTests.txt
aunit.exe    _allPicTests.txt
pause
```

where:
`se` is the Small Edison VM executable program file (`se.exe`) on Windows.
`aunit` is the AUint Tool program file (`aunit.exe`) on Windows.

Here is the corresponding output generated by the batch file above in validating that the suite of all (15) tests have passed and are OK:

[1]The file extension `.pic` means a (binary) position independent code.

```
C:\>type         _allPicTests.txt
Test f0
0
0
Test f1
0
0
Test f2
2350
2350
Test f3
falsetrue
falsetrue
Test f4
15-150truefalsetruefalsetrue
15-150truefalsetruefalsetrue
Test f5
truetruetruetruetruetruefalsefalsefalsefalsefalsefalsefalse9-9
truetruetruetruetruetruefalsefalsefalsefalsefalsefalsefalse9-9
Test f6
12
12
Test a0
4018018
4018018
Test a1
A#
A#
Test a2
[falsetruefalse]
[falsetruefalse]
Test a3
31A2B3C
31A2B3C
Test c0
A#
A#
Test c1
*0true#35
*0true#35
Test m0
313
313
Test r0
12
12
C:\>aunit.exe    _allPicTests.txt
AUnit v1.0.9e.20201103
Copyright (C) Michel de Champlain 2006-2020.  All Right Reserved.
...............
15 tests OK
```

Task 1 [3%]: Complete the AUnit tool on Arduino Nano

This AUnit tool on Arduino Nano should implement a tiny shell using your BSL Uart polling component based on the following header file. You may have done it already in Lab 5 if you have completed the implementation of the `bsl_Uart_TxChar()` function:

```
// bsl_Uart.h - Uart header for Arduino Nano

#ifndef __bsl_Uart_h
#define __bsl_Uart_h

void bsl_Uart_Init(void);
void bsl_Uart_TxChar(char c);
char bsl_Uart_RxChar(void);

#endif
```

You have to complete the implementation of two files:

1. `TestAUnit_task1_todo.c` and
2. `COutForAUnit_task1_todo.c`

```
// TestAUnit_task1_todo.c - Test AUnit for Arduino Nano.

#include "COutForAUnit.h"
#include "bsl_Uart.h"

#include <stdint.h>
#include <stdbool.h>

static void Test1(void) {                // This test will be a success (.)
    PutS("Test 1 - Test Number one\n");
    PutS("1\n"); // Expected output
    PutS("1\n"); // Current output
}
static void Test2(void) {                // This test will be a failure (F)
    PutS("Test 2 - Test Number two\n");
    PutS("23\n"); // Expected output
    PutS("24\n"); // Current output
}

typedef void (*TestEntry)(void);

#define TestMax  4  // Up to 9.

static TestEntry  tests[TestMax] = {
    Test1,
    Test2,
//  Test3,
//  Test4,
//  ...
```

```
//  Test9
};
/*-------------------------------------------------------------------------
 * main
 *-----------------------------------------------------------------------*/
int main(void) {
    bsl_Uart_Init();

    bool testRun = true;

    PutS("Test AUnit on Arduino Nano v1.0\n");
    PutS("Usage: Enter <n> where n is the test number 1..");
    PutX4(TestMax); PutS(" or '0' (zero) to quit.\n");

    while (testRun) {
        PutS("$ ");
        char cmd = GetC();

        // Your code...

    }
    PutS("bye!\n");
    return 0;
}
```

Where `TestMax` can be modified up to nine (9) as the maximum number of tests you can added in this current implementation. Here is an example of an execution of your AUnit tool for Arduino Nano when your implementation is completed:

```
Test AUnit on Arduino Nano v1.0
Usage: Enter <n> where n is the test number 1..4 or '0' (zero) to quit.
$ 1
Test 1 - Test Number one
1
1
.
$ 2
Test 2 - Test Number two
23
24
F
$ 3
Test '3' not referred.
$ 4
Test '4' not referred.
$ 5
Invalid test number. It should be 1..4 or '0' (zero) to quit.
$ 0
bye!
```

Task 2 [3%]: Do a small shell to dump memory on host and target

This task is done in two steps and must reuse your `COutForAUnit.c` and `bsl_Uart.c` implementations (from Task 1).

Step 1: Write a dump memory function based on the following interface:

```
// DumpMemory.h - Dump Memory Function Interface

#ifndef __DumpMemory_h
#define __DumpMemory_h

#include <stdint.h>

void DumpMemory(uint8_t* from, uint16_t nBytes);

#endif
```

You should use the following test program (given) to test you function on host and target:

```
// TestDumpMemory.c - Test Dump Memory Function for Host and Arduino Nano.

#include "DumpMemory.h"

#if !defined(Host)
#include "bsl_Uart.h"
#endif

/*---------------------------------------------------------------------
 * main
 *---------------------------------------------------------------------*/
int main(void) {
    static uint8_t mem[] = {
         0,  1,  2,  3,  4,  5,  6,  7,  8,  9,
        10, 11, 12, 13, 14, 15, 16, 17, 18, 19,
        20, 21, 22, 23, 24, 25, 26, 27, 28, 29,
        30, 31
    };

#if !defined(Host)
    bsl_Uart_Init();
    DumpMemory( (uint8_t*)0, 32); // 32 registers on Nano.
#endif

    DumpMemory(mem, 32); // Based 'mem' address (0x0100) on SRAM data.

    return 0;
}
```

Step 2: Write a small shell to dump the 32 memory registers (from address 0x00 up to 0x31) on target. The execution of your shell should be like this:

```
Shell Nano v1.0
Usage: type 'm'(memory) and 'q' to quit.
$ m
0000 00 00 D3 50 53 49 20 52 56 41 01 10 0C 00 20 00
0010 00 00 01 00 90 50 20 00 20 60 57 01 20 00 1F 00

$ s
Invalid command.
Usage: type 'm'(memory) and 'q' to quit.
$ q
bye!
```

Task 3 [8%]: Do a first VM port on your host development platform

Do a first VM port on your host development platform (Windows, Mac, or Linux) using your preferred C/C++ compiler and validate/test your implementation by running the suite of pre-compiled programs [1][2][3][4][5]. Implement a file loader to read a `.pic` file, which contains one integer (as an array of characters) per line. The last line contains the integer -1.

Make sure:

- To have an independent console output interface that will be reused by the same virtual machine (on the host and target). Use a factory to isolate the configuration.
  Example: No `printf`, but rather functions in the COut such as PutC, PutS, PutI, PutX4, etc.
- To make code of the virtual machine as portable as possible. Use the C standard types with `stdint.h`
  Example: No `int`.

Demo 1 [4%]: First demo and code review

On Wednesday, November 17th, there will be your first (kind of "sprint") demo and code review during lecture time. **All team members must be present, with no marks for the absent member. There will be 15-20 minutes per team. Each task explanation should be taken care of by only one member.**

The first demo will be in the following order: Team 1 (17:45), Team 2 (18:05), up to Team 8. The second demo will be in reverse order. So, **all team members, be on time in the waiting room**.

Task 4 [4%]: Do a second VM port on the target

Do a second VM port on the Arduino Nano target board using one or two precompiled programs [4]. Here is an example of code generated in (`f0.pic`) of the program `f0.se` which is stored in the memory array of the VM:

```
int memory[] = {
    443, 1, 1, 0, 0, 406, 84, 474, 406, 101, 474, 406, 115, 474, 406, 116, 474, 406, 32
    474, 406, 102, 474, 406, 48, 474, 476, 406, 48, 474, 476, 406, 0, 473, 476, 415, 412, -1
};
```

Task 5 [9%]: Replace the host loader with a target serial loader

Replace the (host) memory loader with a (target) serial loader using the UART to load programs in SRAM memory and validate your Arduino Nano implementation by loading and running the suite of pre-compiled programs [4]. This should be done in two steps:

1. Exercise your target serial loader in isolation (without the VM) by a simple test loader (`hal_TestLoader.c`).
2. Integrate your functional target serial loader with the VM.

Here is the Loader interface (`hal_Loader.h`):

```
// hal_Loader.h - Loader Interface

#ifndef __hal_Loader_h
#define __hal_Loader_h

#include <stdint.h>

#define Ack ((uint8_t)0xCC)
#define Nak ((uint8_t)0x33)

#define PacketSizeMax ((uint8_t)11)

typedef enum {
    Success = 0x40,
    // Errors during CheckPacket():
    UnknownCmd,
    InvalidCmd,

    // Errors before command execution:
    InvalidAddr,

    // Errors after command execution:
    MemoryFailed,
    ChecksumInvalid
} Status;

typedef enum {
    CmdBegin = 0x20,
    Ping = CmdBegin,
    Download,
    Run,
    GetStatus,
    SendData,
    Reset,
    CmdEnd
} Cmd;

uint8_t hal_Loader(uint8_t* mem);

#endif
```

Here is the basic structure of the (hal_TestLoader.c):

```
// hal_TestLoader.c - Test Loader implementation for HAL using bsl_Uart.c

#include "bsl_Uart.h"
#include "hal_Loader.h"
#include "vm.h"

...

#define Target      "(ATMega328P)"
#define VMName      "Small Edison Virtual Machine "
#define AppSuffix   ""
#define AppName     "se"
#define Version     " v1.0 "
#define Copyright   "Copyright (c) 2021  Michel de Champlain"

// Banner = VMname AppSuffix Version Copyright
static void DisplayBanner() {
    PutS(VMName); PutS(AppSuffix); PutS(Version); PutS(Target); PutN();
    PutS(Copyright); PutN();
}

#define MemMax 36

static uint8_t  mem[MemMax];

int main() {
    bsl_Uart_Init();
    hal_Init();

    while (true) {
        uint8_t status = Success;

        if ( (status = hal_Loader(mem)) == Success ) {
            DisplayBanner();
            VM_Init(mem);
            VM_execute(mem);

            // Send an Ack to tell the Host that program's execution is done.
            PutC((char)Ack);
            PutC((char)0);
        } else {
            PutS("Loader error: "); PutX4(status); PutN();
            return -1;
        }
    }
    return 0;
}
```

As already mentioned, you must use the Lecture 8 SerialLoader.cs in C# on the host side.

Demo 2 [4%]: Second demo and code review

On Wednesday, November 24th, there will be your second demo and code review during lecture time. **All team members must be present, with no marks for the absent member. There will be 15-20 minutes per team. Each task explanation should be shared between members.**

The second demo will be in the following order: Team 8 (17:45), Team 7 (18:05), up to Team 1. **All team members, please be on time in the waiting room**.

Task 6 [5%]: Do your project report

Do your project report and make sure to follow the project report guidelines [9].

Documents provided for this project:

All the documents or zipped files are in the SOEN 422 **Project folder** on Moodle:

```
[1] Document #1 - Small Edison: A Subset of the Edison Programming Language for Arduino Nano
[2] Document #2 - Serial Memory Loader for Arduino Nano
[3] Document #3 - Chapter on The Edison VM Abstract Code (with publisher's permission)
[4] A Suite of precompiled Small Edison programs (including 15 binary files .pic)
[5] All the C source files of the Small Edison Virtual Machine
[6] All binary Java files (.class) of the Small Edison Compiler
[7] All the ‘‘todo" (.c/.h) source files for tasks in the project
[8] Serial Memory Loader on Host (including source file in C#)
[9] Project report guidelines
```

Team Project Submission

Each team is expected to create a `.zip` file that he will upload to Moodle for grading. The following is expected in the archive:

- A folder containing your source code
- A team project report in **.PDF** format, no other formats will be accepted

Create a `.zip` file with the following signature:
Soen422_TeamNumber_Project.zip

Due 11:59 PM - Monday, December 6, 2021