Concordia University
Department of Computer Science and Software Engineering

SOEN422 Fall 2020 Project [50%]

—

Porting a Small 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 the C programming language, called Cm, intended for a restrictive microcontroller environment ATmega368P 8-bit microcontroller with 32K bytes Flash and 2K bytes SRAM. This project will require eight (8) tasks to incrementally achieving the port on target by using the following roadmap:

1. Do a first VM port on the host.
2. Isolate the BSL and HAL layers on the host.
3. Implement and test the BSL and HAL layers on the target.
4. Do a second VM port on the target.
5. Replace the host loader with a target serial loader.
6. Implement an interrupt manager HAL and BSL layers.
7. Implement an I/O device HAL and BSL layers.
8. Do your project report.

Before tackling the tasks, you need to respect some basic rules to fulfill the VM port requirements. I will explain (during lecture time) the details of the requirements for the first critical task: port the VM on your platform. There is a preliminary task that each team must do. You should port the AUnit testing tool to your platform to regress all your tests from day one up to the due dates.

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 dependent. In other words, 90-95% of the C code must be self-tested continuously to ensure that they work well on both platforms.

Task 0: port the AUnit tool to your platform (host)

So the prerequisite to **Task 1** is to port the AUnit tool to their environment (See the source in C, only 141 lines of code). This ANSI version has been working great for decades on Windows. You should recompile this version on Mac and Linux to help you in verifying the VM implementations on your platform. To my knowledge, the only difference in portability might require a conditional compilation in dealing with the end of line (EOL), a slight difference for platforms: an EOL for Windows is `\r\n`; an EOL for macOS and Linux/Unix is `\n`.

Here is an example of my batch file "runCodePatterns.bat" that exercises the cm VM by running each test of the suite (from `T01.exe` to `T12.exe`) concatenating their corresponding output to a text file "Txx.txt," which will be verified by the "aunit" testing tool program.

```
@echo Running Code Patterns Txx...
@if exist Txx.txt  del Txx.txt
@cm T01.exe                    >  Txx.txt
@cm T02.exe                    >> Txx.txt
@cm T03.exe                    >> Txx.txt
@cm T04.exe                    >> Txx.txt
@cm T05.exe                    >> Txx.txt
@cm T06.exe                    >> Txx.txt
@cm T07.exe                    >> Txx.txt
@cm T08.exe                    >> Txx.txt
@cm T09.exe                    >> Txx.txt
@cm T10.exe                    >> Txx.txt
@cm T11.exe                    >> Txx.txt
@cm T12.exe                    >> Txx.txt
type Txx.txt
@echo on
@echo Checking test result...
@aunit Txx.txt
pause
```

where:
`cm` is the Cm VM executable program file (cm.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 (12) tests have passed and are OK:

```
Test 01: Value Types (Literals)
-128|127|127|127|000DECAF|0000AB8D|0|9|a|A|10|10|10|10|10|false|true
-128|127|127|127|000DECAF|0000AB8D|0|9|a|A|10|10|10|10|10|false|true
Test 02: Conditional Operator
3|4|5
3|4|5
Test 03: Bitwise Operators
0000005A|00003C5A|00003C00|FFFFFFA5|FFFFC3A5
0000005A|00003C5A|00003C00|FFFFFFA5|FFFFC3A5
Test 04: Equality Operators
false|true
false|true
Test 05: Relational Operators
true|true|false|false
true|true|false|false
Test 06: Shift Operators
FFFFFFA6|FFFFFFD3|0000F168|00001E2D
FFFFFFA6|FFFFFFD3|0000F168|00001E2D
Test 07: Extended Bitwise Assignment Operators
7FFFFFA6|3FFFFFD3|FFFFFD30
7FFFFFA6|3FFFFFD3|FFFFFD30
Test 08: Prefix and Postfix Operators
7778798887
7778798887
Test 09: if-else Statement
9|0|9|0|1|
9|0|9|0|1|
Test 10: while Statement - countdown
9876543210
9876543210
Test 11: break Statement
9876543210
9876543210
Test 12: Bit functions
|00000000|00000004|00000000|00000004|00000001|00000000
|00000000|00000004|00000000|00000004|00000001|00000000
Checking test result...
AUnit v1.0.9e.20201103
Copyright (C) Michel de Champlain 2006-2020. All Right Reserved.

............
12 tests OK
```

Task 1: Do a first VM port on the host

Do a first VM port on your host development platform (Windows, Mac, or Linux) using your preferred ANSI C compiler and validate/test your implementation by running the suite of pre-compiled programs [1][2][3][6].

Task 2: Isolate the BSL and HAL layers on the host

Isolate the BSL and HAL layers on your host platform. Mainly for the console output module using the printf() function.

Task 3: Implement and test the BSL and HAL layers on the target

Prepare your second port by testing the BSL and HAL layers on the Arduino Nano target board, implementing the UART console output module.

Task 4: Do a second VM port on the target

Do a second VM port on the Arduino Nano target board using "pre-configured and loaded" from basic precompiled programs [4]. Here is an example of a static pre-allocated programs in the VM memory without using the loader:

```
0000 D9 7F    ldc.i8  127
0002 FF 82    trap    82    ; puti
0004 00       halt

static u8 mem[] = { 0xD9, 0x7F, 0xFF, 0x82, 0x00 };
```

Task 5: 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 [2][5].

Task 6: Implement an interrupt manager HAL and BSL layers

Use the trap instruction to add and implement an interrupt management HAL and BSL layers to the VM ATmega368P hardware interrupts. Small programs (will be provided) to validate your implementation. You should follow this interface:

```
// hal_interman.h -- Interrupt Management Interface

#ifndef __hal_Interrupt_h
#define __hal_Interrupt_h

#include "_stdtype.h"

void  hal_Interrupt_Disable(void);
void  hal_Interrupt_Enable(void);
u16   hal_Interrupt_SaveAndDisable(void);
void  hal_Interrupt_Restore(u16 flags);
void  hal_Interrupt_SetVector(u8 number, u32 handlerAddr);
u32   hal_Interrupt_GetVector(u8 number);

#endif
```

Task 7: Implement an I/O device HAL and BSL layers

Use the trap instruction to add and implement an I/O device HAL and BSL layers to the VM to the ATmega368P I/O port registers. Small programs (will be provided) to validate your implementation. You should follow this interface:

```
// hal_ioreg.h -- IO Register Interface

#ifndef __hal_IOReg_h
#define __hal_IOReg_h

#include "_stdtype.h"

u32  hal_IOReg_Read (u32 ioreg);
void hal_IOReg_Write(u32 ioreg, u32 value);

#endif
```

Task 8: Do your project report

Do your project report [7] and prepare your VM to be tested in a Git repository.

Documents provided for this project:
[1] The Cm Virtual Machine Instruction Set for Small-footprint Embedded Systems
[2] Code Patterns: A Suite of Precompiled Programs for the Cm VM (including 12 binary files .exe)
[3] All the C source files of the Cm VM.
[4] Preconfigured and loaded basic precompiled programs (hardcoded in static C arrays)
[5] Serial Memory Loader (including source files in C)
[6] AUnit - A Simple Language-agnostic Tool for Automated Unit Testing (including source file in C)
[7] Project report guidelines