

Raspberry Pi Embedded Operating System and Runtime

by

James J. Perry

In Partial Fulfillment of the Requirements for the Degree of

Master of Science

In

The Department of Computer Science

State University of New York

New Paltz, New York 12561

February 29, 2016

Raspberry Pi Embedded Operating System and Runtime

James Perry

State University of New York at New Paltz

We, the thesis committee for the above candidate for the
Master of Science degree, hereby recommend
acceptance of this thesis.

Hanh Pham, Thesis Advisor

Department of Computer Science, SUNY New Paltz

Chirakkal Easwaran, Committee Member

Department of Computer Science, SUNY New Paltz

Michael Otis, Committee Member

Department of Engineering Programs, SUNY New Paltz

Approved on _____

Submitted in partial fulfillment of the requirements
for the Master of Science degree in
Computer Science at the
State University of New York at New Paltz

Table of Contents

1.0 Introduction

2.0 Alternative Solutions and Related Work

2.1 Other Raspberry Pi Operating Systems

2.2 IoT Support and Protocols

3.0 The Problem

3.1 Supported Environment

3.1.2 Development Environment

3.1.3 Customization Environment

3.1.4 Target Run-time Hardware Raspberry PI

3.2 Processes and Threads

3.3 Memory

3.3.1 Pre-boot Memory Allocation

3.3.1.1 General memory allocation

3.3.1.2 Process memory allocation

3.3.1.3 Segmentation and Virtual Memory Support

3.4 GPIO

3.5 Graphics

3.6 File Storage

3.7 Programming Environment

3.8 Installation and Customization

3.8.1 Hardware Definition

3.8.2 Operating System Feature Selection

3.8.3 User Program Assembly/Build Support

3.8.4 Image Build Support

3.8.4 Support Not Required

4.0 The Solution

4.1 Supported Environment

4.1.1 Development and Installation/Customization Environment

4.1.2 Run Time Environment

4.1.2 Development Environment

4.1.2.1 Character Support

4.2 Processes and Threads

4.2.1 The Process Scheduling Modes

4.2.1.1 Single Process at a Time Execution (SPE) Mode

4.2.1.2.1 SPE State Transition Table

4.2.1.2 Multi Process Concurrent Execution (MPCE) mode

MPCE State Transition Table

I/O Messaging Execution Table

4.2.1.1 Implications of Messaging code on MPCE Dispatching

4.2.2 The Process Control Data Structures

4.2.2.1 The PCB

4.2.2.2 Process State Queues

4.2.3 Inter-process Communication

4.2.3.2 Writing to a pipeline (msgWrite)

4.2.3.3 Reading from a pipeline (msgRead)

Figure 4.9 msgRead()

4.2.4 Initial Process Scheduling and Scheduler Start

4.3 Memory

4.3.1 Address Protection

4.4 GPIO

4.4.1 GPIO Support List

4.5 Graphics

4.6 Programming Support

4.8 Installation and Customization

4.8.1 The Dashboard

4.8.1.2 Customization

4.8.1.2.1 Hardware Configuration

4.8.1.2.1 Operating System Configuration

4.8.1.2 Build

4.8.1.2.1 Assembly Tools

4.8.1.2.1 Build Status

4.8.2 Customization Processing

5.0 The Analysis and Experiments

5.1 Experiment 1: OS Footprint—Using ARM Assembly Language

5.1.1 Description and Rationale

5.1.2 Evaluation

5.2 Experiment 2: Customization--Footprint

5.2.1 Description and Rationale

5.2.3 Evaluation

5.3 Experiment 3: Processes and Memory Segmentation

5.3.1 Description and Rationale

5.3.2 Evaluation

5.4 Experiment 4: Atomic Message Execution

5.4.1 Description and Rationale

5.4.2 Evaluation

6.0 Conclusion

6.1 Observations and Lessons Learned

6.1.1 Small Footprint

6.1.2 Complex Atomic Instructions

6.1.3 Software Engineering

6.2 Final Thoughts

7.0 Appendix II, Abbreviations and Glossary

8.0 Overview of the rEOS Virtual Machine Runtime

8.1 rEOS VMR Design Overview

8.1.1 Virtual Registers and Memory

8.1.2 rEOS VMR Wordcodes

9.0 Source Code

9.1 rEOS ARM Assembly Source Code

9.1.1 coregraphics.s

9.1.2 font.s

9.1.3 gpiokled.s

9.1.4 gpiopin.s

9.1.5 main.s

9.1.6 mpce.s

9.1.7 processmgr.s

9.1.8 processmsg.s

9.1.9 string.s

9.1.10 time.s

9.1.11 util.s

9.1.12 vid.s

9.1.13 vm.s

9.2 Customizer Source Code

9.2.1 ConstTable.java

9.2.2 CustConfig.java

9.2.3 GPIOConst.java

9.2.4 REOSCustomizer.java

9.2.4 REOSInitFactory

9.2.5 REOSJVMAssm

9.2.4 SymbolTableEntry

9.3 Samples of customizer generated code
9.3.1 config.txt

9.3.2 init.s (SPE)

9.4 rEOS VMR Program Examples

9.4.1 ostester

10.0 Bibliography

1.0 Introduction

This thesis explores the creation of a small footprint, high-performance Embedded Operating System (EOS)--for the Raspberry Pi (RPi). Using a customization approach, the image is configured to include only required functions and omits non-essential functions. The result preserves available memory and storage for use during runtime of an embedded solution. As part of this process, the thesis leverages the resulting run-time environment to provide complex functions (i.e. inter process messaging and GPIO support) that run atomically (non-interruptible). Because of its low cost and power requirements, the RPi is a sound hardware base for embedded systems; however, the generally available RPi operating systems focus on personal computing, dedicated device (i.e. media players), and web serving application (http servers, Minecraft ® servers). The Raspberry Pi reduced Embedded Operating System (rEOS) created by this thesis addresses the EOS support needed to bring the RPi fully into the embedded arena. The rEOS is a bare metal operating system built using the GNU ARM assembly toolchain in Eclipse as a set of assemble-able and linkable modules selected as input to a Java Application running on a Windows personal computer.

The rEOS solution differs from common LINUX solutions in its narrowed purpose and scope (in contrast to LINUX's general purpose and scope), its customization at the micro-system and subsystem level (in contrast to LINUX's macro or system level customization), its small footprint, and its unique Virtual Stack Machine intended to support the creation of programs that address the special needs of this problem set. This Virtual Machine supports a custom

assembly-style programming language that supports multi-process programming based on customization, and a Virtual CPU process scheduler.

Section 2 explores alternate solutions and existing related work in the problem space. This section contains a brief foray into the IoT environment as a background to future (post thesis) development efforts. The nuances of the EOS problem in the context of the RPi are shown in Section 3, which results in a general outline of requirements for the rEOS. Section 4 contains a detailed look at the design and implementation of the rEOS to address Section 3's requirements. While this thesis focused primarily on the operating system design, much of the functionality described has been implemented as a proof of concept, and can be found in Section 9. Section 5 describes two applications that demonstrate rEOS design. Section 6 discusses the particular findings of thesis. Sections 7 through 10 contain supporting material for the project. In order, these sections contain: a glossary, the overview of the custom rEOS virtual machine language, source code and a bibliography.

2.0 Alternative Solutions and Related Work

Before embarking on this project, multiple, existing Raspberry Pi operating systems were evaluated. Additionally, it considers several IoT requirements for inclusion into future (post-thesis) releases and the reasons for excluding them from this project.

It is important, before considering the alternate solutions and related work, to make a distinction between an embedded system and an embedded operating system (EOS). An embedded system has limited user and application-programming interfaces and is a “microprocessor based system that is built to control a function or a range of functions and is not designed to be programmed by the end user in the same way a PC is” (Heath 2013). Such a system does not support the running of user programs, but instead provides a specific—often hardware—interface to the user.

An EOS might simply be defined as an operating system for an embedded system. Such a definition obscures both the value and complexity of an EOS. The term EOS is sometimes considered a form of a Real Time Operating System (RTOS), because of the requirement for both to respond to user input in a timely fashion. While quick response is important in the rEOS, some applications may execute for extended periods of time without direct input. Therefore, in this thesis, EOS is used to mean an efficient, operating system with small memory and storage footprints customized for a specific purpose at installation time that abstracts hardware interaction through a specifically defined set of operating system and software support.

2.1 Other Raspberry Pi Operating Systems

Much of the popularity of the RPi stems from the large number of (largely LINUX-based) operating systems, each with support for a particular problem space. Some of these OS address very specific needs, so they do not provide viable solutions for the EOS requirement of the rEOS. Other available OS lack either support for the Broadcom System on a Chip (SOC) or lack the full functionality and flexibility required.

OpenElec is a custom Linux-based operating system written to support KODI, formerly known as XBMC (a free and open source media player to provide Xbox Media Center-like support). The media specific focus precludes it from consideration for this project. RaspBMC is a similar OS, with similar limitations, to OpenElec.

The most popular RPi OS is Raspbian, which is an RPi specific implementation of the popular Debian Linux OS. Both are fully functional, personal computing operating systems with relatively large footprints. As such, they have standard file-system support, support real-time interactions with users, and are programmable in popular shell and scripting languages such as Bash and Python. However, the Raspbian image strains the limit of memory, storage and CPU resources. It provides very limited support for customization—the paradigm being to extend the base OS image with software, rather than determine or limit the OS support at build time. Finally, Raspbian's interaction with the RPi GPIO functions is most easily accomplished through a scripting interface. As such, Raspbian is a fine solution given limited resource demands, but it lacks support or tools for more demanding autonomous and GPIO-centric applications.

BareMetal supports multi-core processors (which the RPi does not have) and is INTEL/AMD based. Thus it cannot be used.

The eCos OS is perhaps the closest match for the needs of this project. It is a customizable OS that can generate a small footprint image containing only the required subsystems. Despite eCos's support for some ARM processors, it does not natively support the RPi's Broadcom SOC.

2.2 IoT Support and Protocols

I considered positioning the rEOS to provide support for the Internet of Things (IoT). Ultimately, future releases of the rEOS would likely add such support. Internet services require some sort of network service. The RPi hardware configuration could host a full IP Stack, however much of the support provided is not needed to make simple requests to RESTful services. Recognizing this and other issues associated with resource-constrained embedded systems, the IETF's Constrained Application Protocol (CoAP) defines a lighter weight interface allowing requests over UDP. Supporting network connectivity would require the rEOS to supply support for hard-wired access via the networking port or to support a USB connected wireless dongle. Either effort requires significant operating system support.

The use of the RPi built-in 10/100 mbps Ethernet port requires less OS support than the USB, since it is networking only. It only requires networking and TCP/IP stack. Adding support for a wireless USB network dongle requires both a sizable amount of code for the USB interface and then still requires the networking code. Although adding a USB interface is attractive because it could be used to support a USB keyboard, mouse, or Flash drive, general purpose

USB support is not only programmatically complex, but requires fairly large memory and storage to support. Given these considerations, IoT support is not included in rEOS.

3.0 The Problem

An Embedded Operating System (EOS) installed on the RPi and intended for use in complex embedded systems, autonomous robots, and gaming requires certain key features to allow the greatest use of limited, available hardware resources. Generally, these problems involve interaction with electronic circuits and components through the RPi GPIO support, strain the memory and storage capacities of the hardware and leverage a runtime environment that supports user defined programs and support for multiple process management. Only limited support for graphics is required for most applications, but supporting a range of color depth and pixel resolution allows for a greater variety of displays, especially for use in a gaming environment. The runtime environment must provide programming language support. Additionally, the embedded system must provide customization and installation support that abstracts the configuration and installation of the EOS onto the device from the user.

Customizers, installers and builders of rEOS systems must be competent information specialists or computer scientists with the skills to install and test all the tools used in the solution. Ideally, they would have a basic knowledge of:

- Java Programming
- ARM Assembly,
- Cross Compilation in an IDE,
- Command line Operating System commands and “power-user” knowledge of the build machine operating system.

- Familiarity with the Raspberry Pi, the Broadcom 85620 SOC, and the ability to read, understand and apply electronics and computer architecture reference materials, including block level schematics.

The subsections of section 3 explore each of these general requirements and enumerate the specific design requirements of the solution.

3.1 Supported Environment

The Raspberry Pi suite of offerings currently includes multiple complete systems (including the Raspberry Pi 1 Model A+, Raspberry Pi 1 Model B+ and Raspberry Pi 2 Model B, Raspberry Pi 3) and the Compute Module, which requires the Compute Module Development kit. The Raspberry Pi Model B is also readily available for purchase in retail markets. These various modules provide different hardware resources and require custom operating system support; however, that support is largely a matter of address configuration.

In addition to the target run-time hardware, the project requires a platform on which to support the actual solution development and a place to customize and build the installable rEOS operating system image.

3.1.2 Development Environment

The development environment might be built on either a Windows or Linux platform, but in either case it must supply:

- all of the software engineering tools including a cross-compilation IDE,
- support for SD Card reading and writing,
- and a networking connection.

3.1.3 Customization Environment

The customization environment might be built on either a Windows or Linux platform, but in either case must supply:

- all of the software engineering tools including a cross-compilation IDE,
- support for SD Card reading and writing,
- Java Runtime Environment (JRE 7.0 or later).
- and a network connection.

3.1.4 Target Run-time Hardware Raspberry PI

Ideally, this thesis solution would support multiple or all Raspberry Pi models. Doing so, however, requires significantly more hardware resources (and therefore money) and testing time. Since the various models differ in configuration, most support would take the form of different GPIO pinouts and corresponding memory addresses for the hardware. This type of support might easily be added as a customization function in a later release, and therefore it is tabled until then. This project must support only the Raspberry Pi Model B.

3.2 Processes and Threads

The rEOS process management must supply a basic process life cycle that includes new, waiting, ready, running, terminated states. The diagram below shows the state transitions as described in (Silberschatz 2014). It should be noted that interrupt stimulus that triggers the transition from running to ready in the diagram will be handled by the rEOS VM via an instruction count threshold rather than a time slice or hardware interrupt.

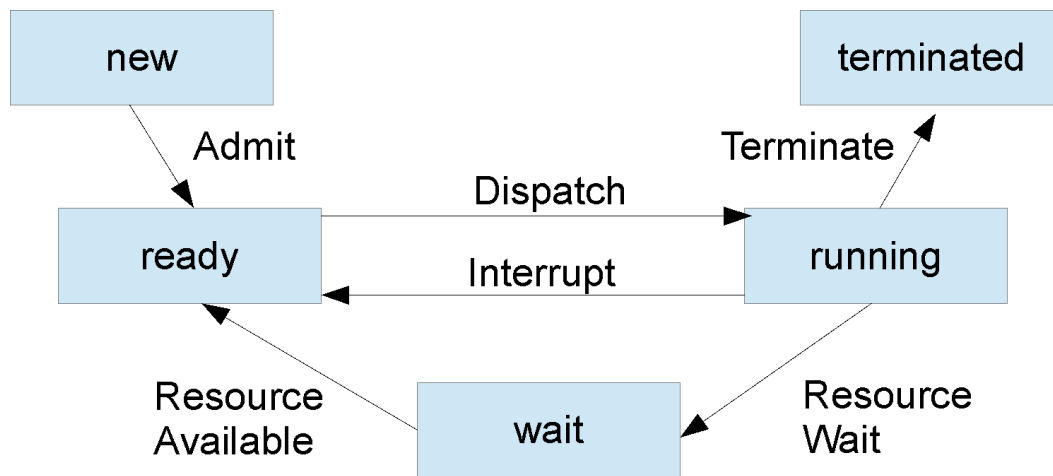
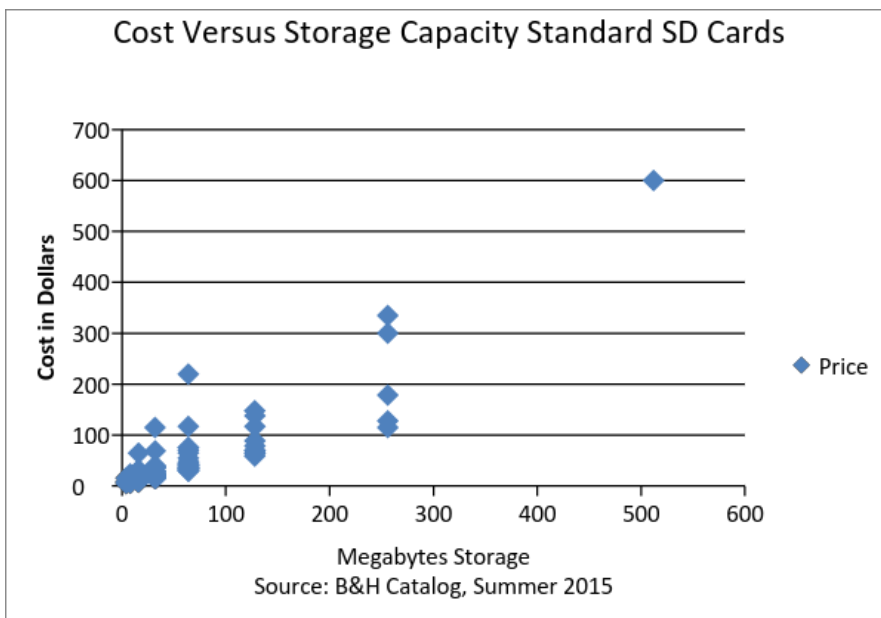


Figure 3.1 General Process States

3.3 Memory

A key problem for rEOS is the limited memory and storage resources for the RPi. The SOC memory of the Raspberry Pi 1 B is 512MB. This memory is allocated to both the CPU and GPU, depending on a configuration setting. Any operating system code must reside within this footprint. While storage capacity can be addressed in part by the purchase of higher density SD cards, the cost of a 32GB SD card is greater than that of the RPi unit itself.



3.3.1 Pre-boot Memory Allocation

Among other things, the RPi boot process loads the file `config.txt`. This file contains a setting to determine the memory allocation split between the GPU and the CPU. After the boot process, this setting cannot be changed without rebooting the system (Perry 2014).

The memory allocation support required for the rEOS is:

- A future addition to the customizer tool will generate a suggestion for the allocation split between the CPU and GPU. This suggestion can be computed based on the choices made during the other parts of the customization process. This requires updates to the `config.txt` file.
- A method that allows the user to set the CPU/GPU allocation

3.3.1.1 General memory allocation

All code (operating system and user) must run within the memory allocated to the CPU. This includes the portion of the rEOS that must always be resident, the stack machine, the user code, and the data areas associated with these processes. A future customization tool must determine the minimum memory required for these processes and data and ensure that sufficient resources are reserved during the customization process for the initial processes. In other words: the future CPU/GPU user memory customization must enforce a minimum memory allocation and not allow users to under allocate the CPU memory at startup. Subsequent processes are handled by the scheduler. Note: the customization process should consider that CPU memory must be at least large enough to handle the rEOS core code and the resident pspace of all processes defined at customization.

3.3.1.2 Process memory allocation

The resource constrained nature of the RPi as an EOS requires that processes be allocated resources that are both sufficient and frugal. To fulfill this requirement, and to allow flexibility, two approaches must be addressed: predetermined resource boundaries (determined during customization) and runtime process resource allocations.

During the customization process, the user defines a stack (data) memory requirement for each process that will run on the system.

3.3.1.3 Segmentation and Virtual Memory Support

Often, the memory available for a process is not available in actual memory. Moreover, it is well known that most programs do not require access to all their variables or code at all times (Siberschatz 2014).

The rEOS must support segmentation by dividing all programs into program (pspace) and data/stack (sspace) spaces. This thesis allows only one copy of each process, but a simple updates the customization design would allow multiple copies of a process to the same pspace during execution and have unique copies of their own sspaces. This segmentation takes best use of the RPi memory space, while allowing each process to maintain its own data.

Virtual Memory is an important future release consideration that must support processes upon admit or dispatch. It must be noted that this support does not require process execution time allocation and deallocation, such as that provided by the C language `stdlib.h` functions: `malloc`, `realloc`, `calloc`, or `free`. Therefore the requirement for virtual memory is limited to allocation and deallocation of the pspace and sspace spaces during process dispatch and

interrupt. However, it must be noted that the virtual memory management and process management processes must account for the pipeline messaging process and its use of the process stack space for buffers.

3.4 GPIO

The Raspbian OS support for the GPIO pins of the RPi is generally through a scripting language using a bash set command or a Python command. Users must analyze the GPIO pins that they used and create scripts to support them. The rEOS must simplify this work by creating a simple interface to the GPIO that supports:

- Aliases for pins,
- Easy read/write access,
- Polling or waiting through the reattempt of the i/o instruction by the user program.

3.5 Graphics

Graphics must be supported through the GPU and mailbox scheme described in the BCM 2835 ARM Peripherals documentation. Briefly, the interface is a read/write mail box that provides a way to provide and write to a Frame Buffer. The technology could be exploited to provide a standard single-buffer graphics interface or to leverage double buffering for more graphics intensive applications.

3.6 File Storage

Future versions of the rEOS must provide lightweight file and file system support.

3.7 Programming Environment

The programming support must provide a higher than ARM Assembler instruction set in the form of an imperative programming language that abstracts the underlying hardware interfaces and file system operations.

These instructions must supply stack architecture support that includes arithmetic operations, logical operations, execution flow control, gpio interaction, process start and fork operations (in a follow-on project), and inter-process messaging.

3.8 Installation and Customization

The installation and customization process must provide several core functions: a GUI Interface that simplifies the definition of the target RPi hardware configuration, the selection of operating system features, a build and assembly of user defined programs, and the creation of a bootable Operating System disk image that can be transferred to a supported SD card for use in the RPI.

3.8.1 Hardware Definition

The creation of the rEOS Image depends in part on identifying several hardware features to the build process. These features must of course actually exist on the target RPi, but it is possible to exclude support of some hardware features to minimize the disk footprint of the rEOS image and to speed-up execution.

The customizer must support customization of:

- the inclusion or not of GPIO pin support,
- the inclusion or not of GPIO OK LED support,

- the graphics color density,
- the display mode
- the SD card size.

3.8.2 Operating System Feature Selection

The rEOS must provide basic operating system features such as dispatching processes, memory management, etc., but other features should be included, only if required by the EOS.

The rEOS must support means to:

- choose a sequential or a multitasking process dispatch mode,
- include or exclude virtual memory support (disabled in this release),
- include file i/o support (disabled in this release),
- include inter-process messaging,
- draw simple graphics (lines, dots, fill rectangles),
- manipulate and display strings
- determine the abend or not on unsupported instruction, or user initiated dump,
- specify first-fit or best-fit memory allocation methods (only resident programs exist in the proof of concept release).

3.8.3 User Program Assembly/Build Support

The operating system build environment must allow users to write and include their own programs in the operating system image. These programs consist of assembly-like mnemonics for the wordcodes supported by the virtual machine that are assembled into ARM .s assembly and included in the Image Build Support.

The customizer must:

- allow the user to specify the source path,
- allow the user to specify the source file (in accordance to a naming convention),
- allow the user to specify a data space size,
- allow the user to assemble the program from the GUI Interface,
- allow the user to define the resulting process as resident.
- display the partially complete PCB after assembly.

3.8.4 Image Build Support

The customization build support must evaluate all the input parameters, incorporate all user programs and create a complete system image ready to be copied to an SD card and used as an EOS. It must also display an inventory of all files included in the image after the build.

It is not required that the builder tool ensure that all the image include all support required by instructions needed in user programs. This allows for the partial execution of programs during development.

3.8.4 Support Not Required

Non-computer scientists often equate operating systems with the functionality commonly provided by a personal computing operating system like Windows or Apple OS. The EOS problem space usually requires a different set of operating system support. The following representative (but hardly comprehensive) list of functions not required by this project serves to clarify the project goals:

- Keyboard Support,

- Mouse Support,
- Network Communications,
- USB Adapters,
- RPi camera,
- File Directory navigation,
- File editing,
- User Interface via GUI or command line,
- Office tools,
- Software not run in the rEOS VM.

4.0 The Solution

The rEOS fulfills the requirements of the previous section with a three-faceted design, shown in the diagram below.

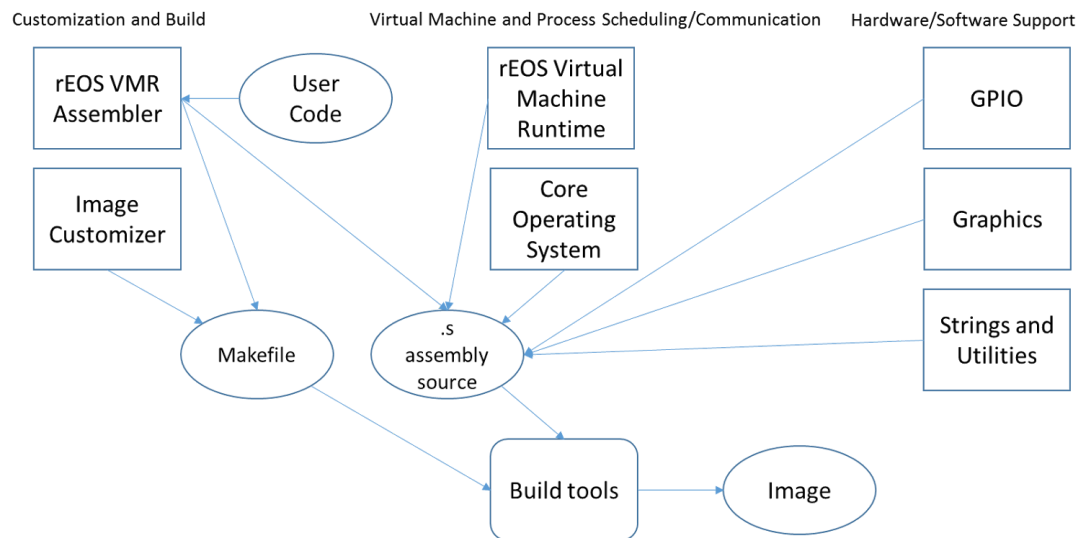


Figure 4.1 Overall rEOS Solution

The core of the solution design is the rEOS Virtual Machine Runtime (rEOS VMR) and the core operating system. The rEOS VMR provides the run-time programming environment, where user programs are executed. The core operating system schedules processes and manages the rEOS VMR, and it manages and interacts with the system resources (memory, GPIO, graphics, files).

The core operating system manages these resources through calls to the custom software and hardware functions of the second facet—Hardware/Software Support. These calls are internal to the rEOS and not directly accessible to the end-user programs. Instead, they are leveraged by wordcode instructions through the rEOS VMR.

The remaining facet of the rEOS is the installation/customization and build tools. These tools allow for the creation of user programs and for the customization of the operating system image.

4.1 Supported Environment

The rEOS solution consists of an installation/customization environment which runs on a Windows/PC and the execution environment which executes on a Raspberry Pi B.

4.1.1 Development and Installation/Customization Environment

The development, and installation and customization tools require the same configuration. The tools in that environment are used to edit, compile and configure the rEOS install. The minimum Installation and Configuration System Configuration requires:

- Windows 7
- RAM: 128 MB
- SD Card reader/writer
- Disk space: 124 MB for JRE; 2 MB for Java Update; 1 MB for Arm rEOS source, build, image and object files; 14 MB for Java Customizer code and source..
- Processor: Minimum Pentium 2 266 MHz processor
- JDK/JRE: Java 7 or Java 8
- Editor of plain text files (i.e. NotePad, NotePad++)
- Yagarto, cross-development environment including GNU C/C++ and Eclipse IDE from <http://sourceforge.net/projects/yagarto/>
- MinGW, Minimalist GNU for Windows from <http://www.mingw.org/>

- OS Build Template from Cambridge University Tutorial found at <https://www.cl.cam.ac.uk/projects/raspberrypi/tutorials/os/downloads.html>

4.1.2 Runtime Environment

Installation and customization tools are used to edit, compile and configure the rEOS install. The Installation and Configuration System Configuration requires:

- Raspberry Pi B
- SD Card 16 GB
- Hardware as required by application

4.1.2 Development Environment

The development of the rEOS requires the following development environment:

- Raspberry Pi B
- SD Card 16 GB
- Hardware as required by application

4.1.2.1 Character Support

The rEOS supports the lower 128 of the ASCII character set. While fonts may support certain special formatting characters, the rEOS does not provide formatting use of them. These unsupported characters appear blank in the following table.

Like Java, these characters are stored as 16 significant bits in a 32-bit memory space. The loss of compactness is intentionally accepted to allow for double (and greater) byte character sets in a future project.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00																
10																
20		!	“	#	\$	%	&	.	()	*	+	,	-	.	/	
30	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
40	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
50	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_		
60	‘	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
70	p	q	r	s	t	u	v	w	x	y	z	{ }	~			

Figure 4.2 Supported Character Set

At OS build time, fonts can be changed by supplying an alternate font.bin file. may be selected from a list of bitmap files.

4.2 Processes and Threads

The rEOS process model is simple non-hierarchical (that is no parent-child relationships) model with inter-process messages passing through wordcode operations.

REOS process scheduling uses a round robin scheme in one of two modes: either lightweight (and small footprint) single process execution or the more flexible, but more resource demanding, multi-process execution.

4.2.1 The Process Scheduling Modes

The rEOS supports either a single-process-at-a-time execution (SPE) or a multi-process-concurrent execution (MPCE) mode, which is chosen during the customization/build step. The mode is chosen during the build step, so the customizer can initialize certain control data and determine which modules to include in the image build.

When the scheduler admits a new process, it is assigned to the process queue. In turn, each process is placed into a FIFO queue according to the process's status.

4.2.1.1 Single Process at a Time Execution (SPE) Mode

SPE provides a very light weight framework for program execution. Process scheduling consists of processing each process defined in the queue or stack, in order, until all complete execution. All process memory is allocated in memory during initialization. No process is interrupted during execution and no changes in the priority are allowed after build. The impact of this design is that a series of programs are executed from start to finish, one at a time.

SPE mode process state and state transition model defines a very simple process life cycle (see diagram). When processes are submitted to the Process Scheduler, the process is placed in the new state, and stored in the process queue. When the process queue is empty (that is at initialization or when the current process terminates) the next process in ready is removed from the stack, and its state changed to running. Processes continue in the running state until they exit or abend, when their status changes to terminate, and they are no longer executed by the rEOS VMR. The process queue can be LIFO or FIFO as determined by at customization.

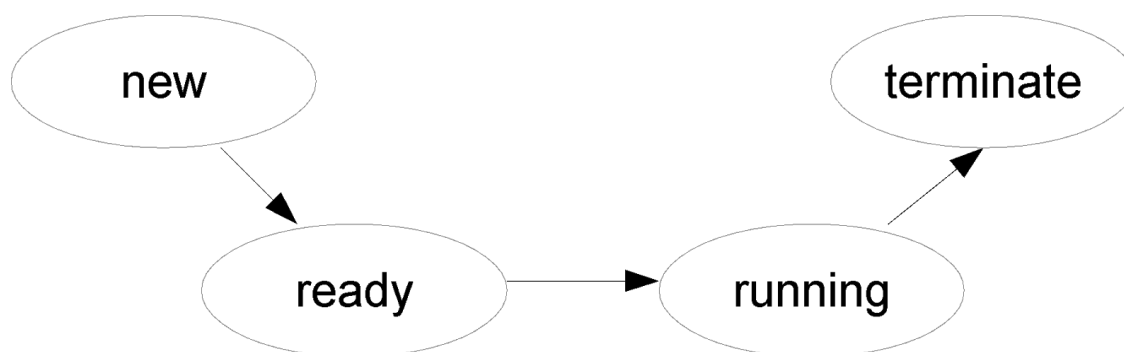


Figure 4.3 rEOS Process States

4.2.1.2.1 SPE State Transition Table

Current State	Stimulus	New State	New Process: New State
New	Admit Process from customization	Ready	n/a
New	Admit Process from fork() (Future Release)	Ready	n/a
Ready	Execute Start	Running	n/a
Running	Abend/Exit	Terminated	n/a
Running	Fork new process (Future Release)	Running	New

Figure 4.4 rEOS SPE State Transition Table

It is important to recognize the ramifications of this simple design and their effect on overall process management design.

Because processes are added to the ready queue—with their required memory resources defined at customization—at operating system boot time only, the SPE mode does not require any memory management support. This allows for a smaller footprint operating system.

4.2.1.2 Multi Process Concurrent Execution (MPCE) mode

MPCE provides a framework for concurrent program execution. Process scheduling consists of processing each process defined in a ring queue in order until all complete execution.

Processes are scheduled one of two ways: at system load, before the invocation of the MPCE scheduler, and (in a future version) during process execution via a `fork()` invocation.

The MPCE mode process state and state transition model offers a modified version of the standard process life cycle (see diagram in section 3.2 and table below). There is no waiting state, because of the rEOS's I/O-Messaging handling scheme. When processes are submitted to the Process Scheduler, the process is placed in the ready state, and stored in the process ring queue. When the process queue is empty (that is at initialization or when the current process terminates) the next process in the ready queue is removed, its state changes to running, and a number of cycles is assigned to the process. Processes continue in the running state until they exit, abend or reach the maximum number of cycles allocated to the burst of execution. If they exit or abend the status changes to terminate, and they are no longer executed by the rEOS VMR.

MPCE State Transition Table

Current State	Stimulus	New State	Child New State
New	Admit Process from customization	Ready	n/a
New	Admit Process from <code>fork()</code>	Ready	n/a
Ready	Execute Start	Running	n/a
Running	Abend/Exit	Terminated	n/a
Running	Fork new process	Running	New

Figure 4.4 MPCE State Transition Table

It is important to recognize the rEOS VMR instructions are atomic (uninterruptable) within the rESO VMR. This is obvious in the simple arithmetic or logic control commands, but less so in the case of the I/O or messaging commands. When these commands are executed, they are either completed or must be retried.

I/O Messaging Execution Table

Mode	Stimulus	New State
SPE	I/O-Message Completes	Running
SPE	I/O-Message Incomplete	Application should loop
MPCE	I/O-Message Completes	Running
MPCE	I/O-Message Incomplete	Application should loop

*Figure 4.5 I/O Messaging Execution Table***4.2.1.1 Implications of Messaging code on MPCE Dispatching**

The messaging protocols described below have implications on process state and life.

Two key issues must be addressed:

- A writing process could have an unread message pending when its target reading process terminates. In this case, `sspace[0]` should be set to `MSG_RC_unwritten`.
- A reading process could have a pipeline open to a writing process with a message pending or `msgOpen` complete with an unwritten message. In this case:
 - if the writer process is in the `PCB_STATE_terminated` the writer should be removed from the MPCE queue.
 - if the writer process is in the `PCB_STATE_ready` or `PCB_STATE_waiting` state the writers `sspace[0]` should be set to `MSG_RC_unwritten`.
 - in all cases, the reading process should be removed from the MPCE Queue

These cases prevent processes from opening messaging but being unable to complete the message.

4.2.2 The Process Control Data Structures

The several data structures required to control and schedule processes include a Process Control Block (PCB), Process state queues and other miscellaneous data.

4.2.2.1 The PCB

The structure of the PCB is a series of word length data items. The following listing shows each word and it's offset from the beginning of the PCB:

```
.equ  PCB_base,          0x00000000 // The start of the PCB
.equ  PCB_pid,          0x00000000 // unique process id
.equ  PCB_state,        0x00000004 // current process state
.equ  PCB_pspace,       0x00000008 // address of pspace
.equ  PCB_psl,          0x0000000C // length of pspace, in bytes
.equ  PCB_sspace,       0x00000010 // address of sspace
.equ  PCB_spl,          0x00000014 // Length of the sspace in bytes
.equ  PCB_lpc,          0x00000018 // lpc
.equ  PCB_lsp,          0x0000001C // lsp
.equ  PCB_cycles,       0x00000020 // VM cycles before swap out.
.equ  PCB_source,       0x00000024 // process read buffer id.
.equ  PCB_target,       0x00000028 // process write buffer id.
.equ  PCB_next,         0x00000032 // Next PCB in Current Queue
```

Several important items must be noted:

The PCB does not have a special field to indicate scheduling mode, because mode is not a process specific attribute. Instead the PBC_cycles field is used to determine how many rEOS VMR instructions to execute before swapping the process out of the running state and to the ready state. However when running in SPE mode, no process is swapped until the running process ends. In this case, PCB_cycles must be set to -1 to indicate that the process should continue to run until an exit instruction or an abend is reached.

There are no child processes in the UNIX/LINUX sense. In future releases a process may fork() a new process, but that process is immediately independent and not a child. Message passing may be set up between the processes, but each receiving process may only have one source process at a time, and each sending process may only send messages to one receiving process.

Only one block of data memory is allocated to the process. If a process requires additional memory while it runs, that cannot be accommodated by the process's sspace, a new process must be forked to complete the action independently.

PIDs are assigned at customization/build time for resident processes.

In SPE mode the quantum is always set to -1, which will cause the process to run to completion without preemption. In MPCE mode, the quantum is set at customization time.

4.2.2.2 Process State Queues

The rEOS uses a single process queue to hold processes ready to be executed. In SPE mode the queue is represented as a singly linked list containing the processes currently being

managed. In MPCE mode, the ready queue is represented as a ring queue. Both are updated just before calling the rEOS VMR to execute a process and immediately upon the return from a rEOS VMR call.

Both queues consist of linked PCB containing a next pointer and the address of a PCB. The scheduler maintains a current pointer points to the next item to process, and a last pointer to indicate the last item process for each queue.

4.2.3 Inter-process Communication

A rEOS process can be independent or cooperating. Cooperating processes “can affect or be affected by the other processes in the system” (Silberschatz 2014). The only interaction between processes in the rEOS is through inter-process messaging, where the rEOS VMR acts as an intermediary. There is no owner/owned or parent/child hierarchy in the rEOS, so no direct interaction between processes exists, except for messaging.

The inter-processing messaging support of the rEOS VMR seeks to minimize the dangers associated with cooperation by providing a means for each process to make a one-way pipeline connection to either send or receive data from another process. A process may have zero or one sending pipelines, and zero or one receiving pipelines. Each pipeline exists for the duration of the msgOpen(), msgRead() and msgWrite() cycles, and then is collected. The life of a pipeline, therefore, is exactly one message.

A single message lifespan for the pipeline simplifies the design of messaging. Moreover, it eliminates the shared bounded buffer problem because the reading process is blocked from reading until the writing process completes its work. Once the write is complete, further

msgOpen() calls for writes by the sending process are blocked until the receiving process completes. Then the pipeline is dissolved. Future interactions require the creation of a new pipeline.

The major drawback of this method is that it does not prevent deadlocks. Because the message pipelines are so tightly coupled and blocking on a pipeline is absolute, a Starving Philosophers deadlock (among other forms) is possible (Singhal 1994).

The next few sections describe the workings of the rEOS VM process messaging. The description assumes that both the sending and receiving process remain active during the message exchange. The handling of processes that terminate during the exchange is described in the MPCE section.

4.2.3.1 Creating a pipeline (msgOpen)

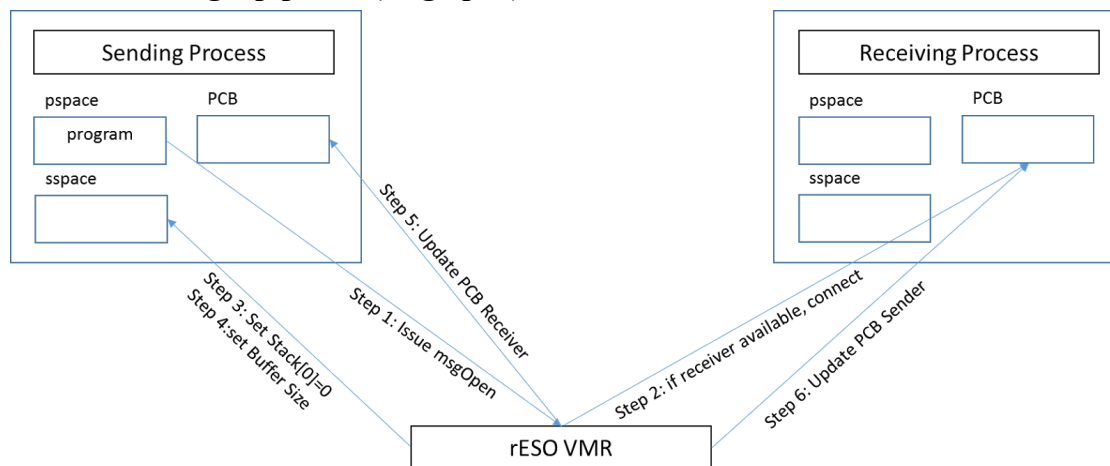


Figure 4.6 msgOpen()

The initial step in inter-process communication is for the sending process to invoke msgOpen with a parameter that identifies the PID of the receiving process. The rEOS VM then checks to see if the connection is viable. If the sending process is not already sending and the

receiving process is not already receiving then the pipeline is created by the rEOS VMR by updating the locking word, setting the buffer size and updating both PCBs. Note: Once the check for viability begins, the operation is atomic until the either: the pipeline is created or the rEOS determines the pipeline is not viable.

4.2.3.2 Writing to a pipeline (msgWrite)

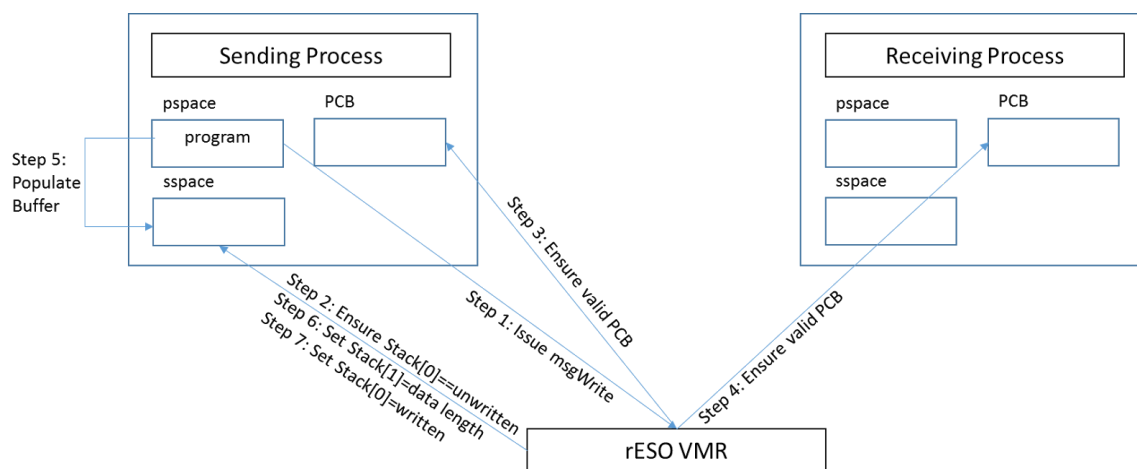


Figure 4.7 `msgWrite()`

Since the pipeline exists to support exactly one, one-way communication, the second step in inter-process communication is for the sending process to invoke `msgWrite`. The rEOS VMR ensures that the pipeline allows writing by checking the locking word and that the respective PCB shows the processes still linked. If not, it fails silently. Next the data and its length are moved into the buffer, and finally the locking word is updated.

This technique protects two processes from being deadlocked because the pipeline suspends the write process blocks the read process until the write is complete, and the read

blocks further write until reads complete. But it is important not to assume that this model is completely deadlock free.

Three processes might easily become deadlocked by having a ring linked set of pipelines. In the diagram below if each process depends on receiving information from its sender before it can send. A deadlock has occurred.

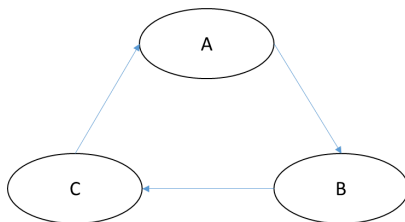


Figure 4.8 Deadlock Diagram

4.2.3.3 Reading from a pipeline (msgRead)

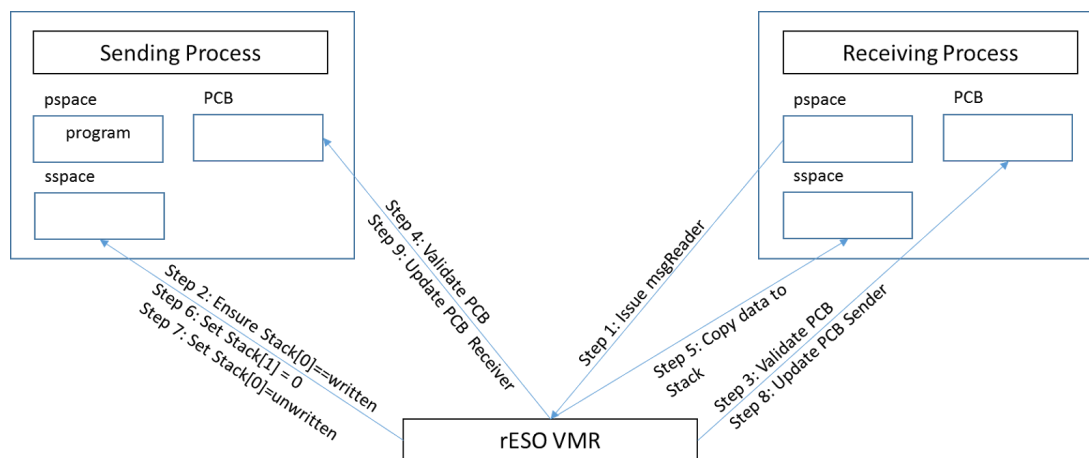


Figure 4.9 msgRead()

The last step in inter-process communication is for the receiving process to invoke msgRead. The rEOS VMR ensures that the pipeline allows reading by checking the locking

word and that the respective PCB shows the processes still linked. If not, it fails silently. Next the data is pushed to the receiving stack, the locking word is updated, and the PCBs updated to break the pipeline connection.

One drawback to this messaging scheme is that the sending process may not begin a new pipeline until after the existing message is read.

4.2.4 Initial Process Scheduling and Scheduler Start

Initial process scheduling and starting the scheduler are handled by the initialization code in the `init.s` module. The sequence of events is:

1. The processing mode is determined during customization and set in the scheduler.
2. The processes, in FIFO order or LIFO order (depending on customization), are defined via the scheduler populate function, which receives a reference to the new process's PCB, updates the PCB to the ready state, and adds the process to the ready queue.
3. After all processes are defined and added in step 2, the schedule start function is called.

4.3 Memory

The memory support for the rEOS VMR provides process memory protection. This is achieved by the rEOS VM during runtime. The design assumes a RPi with 512MB memory. Future rEOS versions can easily be modified to allow memory size as a customizable choice at build time.

4.3.1 Address Protection

Process memory contains two segments: the executable wordcodes (`pspace`), and the data and messaging area (`sspace`). Both memory areas must be separately contiguous and both must

not be “violated by the execution of other processes or the operating system” (Silberschatz 2014). The Process Scheduler ensures this condition on dispatch. This is accomplished through a simple software address scheme based on the common hardware address trap scheme described in (Silberschatz 2014). The trap check happens at the beginning of each wordcode execution cycle.

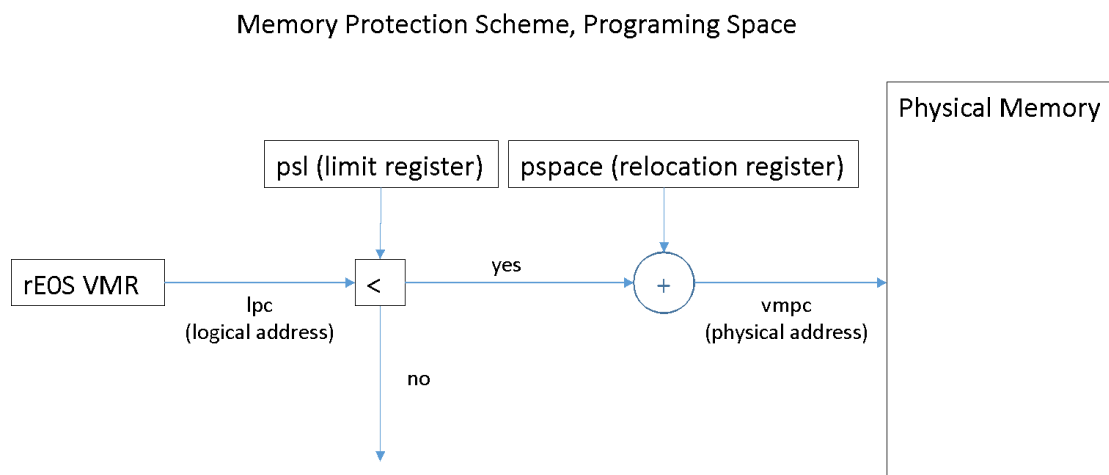


Figure 4.10 Memory Protection Scheme, Programming Space

The Memory Protection Scheme, Programming Space diagram shows the basic plan to protect the pspace. The lpc register holds the offset value of the next instruction to fetch for execution from the beginning of the pspace. It is checked against the size of the pspace insure that the next instruction is within the pspace, and finally the pspace address—the start address of the pspace—is added to create the vmpc—the address of the instruction in physical memory.

It should be noted that this scheme does not explicitly check the validity of the second word of a two-word wordcode. This is an acceptable risk because the assembler ensures that the second byte is within the pspace at assembly/customization time.

The strategy for protecting the data stack space of a process running in the rEOS VMR follows an analogous strategy shown in the Memory Protection Scheme, Stack space with different registers in play.

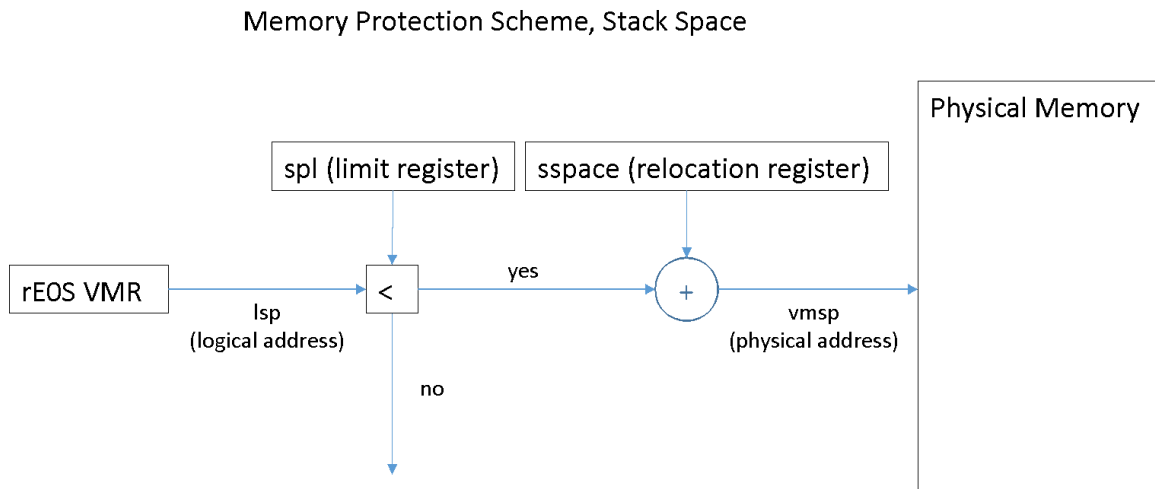


Figure 4.11 Memory Protection Scheme, Stack Space

Like the program space scheme the stack space protections checks at the beginning of the wordcode execution, cycle. The validity check is completed before any pushes to or pops from the stack.

The code needed to always check the stack space pointer is trivial; however it requires performing the check any time the stack pointer changes and so is costly in execution cycles. Thus the risk of corruption must be balanced against faster execution. The key considerations are:

1. The simplest way to check stack safety would be to create separate functions in the rEOS VMR that are invoked to perform push and pop operations and provide the functionality of the stack space memory protection scheme.

2. Not checking might be an acceptable risk since:
 - a. Errors could be checked each wordcode cycle, thus detecting problems after the fact.
 - b. Errors that do not write to other processes have a minimal impact.
 - c. Well-written user processes will not violate stack integrity.

After these considerations, the rEOS VMR will check for stack violations before the fact, and abend any process that violates memory integrity for either the pspace or sspace.

Generally, the stack space checking for rEOS VM wordcodes occurs with each stack space operations. However, certain wordcodes manipulate the top two values of the stack and replace a single position result back to the new stack top. Since the wordcode accesses spaces that have already been checked by previous pushes, the protection check is grouped to enhance performance. In this case, and abend might indicate a previously corrupted stack.

4.4 GPIO

GPIO is supported by the use of reserved words in the rEOS assembler. By using those keywords with the rEOS VMR mnemonics that invokes GPIO functions, the assembler abstracts the interaction with the hardware.

4.41 GPIO Support List

Board Label	GPIO Label	GPIO Pin
OK light	STATUS.LED. N	GPIO 16
GPIO 1	+3v3	power source
GPIO 2	+5v0	power source
GPIO 3	SDA1	GPIO 02
GPIO 4	+5v0	power source
GPIO 5	SCL1	GPIO 03
GPIO 6	GND	ground
GPIO 7	GPIO GCLK	GPIO 04
GPIO 8	TXdP	GPIO 14
GPIO 9	GND	ground
GPIO 10	RXD0	GPIO 15
GPIO 11	GPIO GEN0	GPIO 17
GPIO 12	GPIO GEN1	GPIO 18
GPIO 13	GPIO GEN2	GPIO 27
GPIO 14	GND	ground
GPIO 15	GPIO GEN3	GPIO 22
GPIO 16	GPIO GEN4	GPIO 23
GPIO 17	+3V3	power source
GPIO 18	GPIO GEN5	GPIO 24
GPIO 19	SPI MOSI	GPIO 10
GPIO 20	GND	ground
GPIO 21	SPI MISO	GPIO 09
GPIO 22	GPIO GEN6	GPIO 25
GPIO 23	SPI SCLK	GPIO 11
GPIO 24	SPI CE0 N	GPIO 08
GPIO 25	GND	ground
GPIO 26	SPI CE1	GPIO 07

Figure 4.12 GPIO Support Table

4.5 Graphics

The GPU of the Broadcom SoC on the RPi supports multiple color depths and display resolutions. Support for these various configurations might be provided through additional configuration settings or through a runtime instruction. Instead the rEOS specifies the graphic

configuration during customization. Note: The rEOS does not support double buffering in this release, which limits its application in graphics heavy applications.

The CPU accesses the GPU through a mailbox/postman scheme defined by the BCM2835 ARM Peripherals specification. At a high level, the scheme requires that framebuffer information be shared through the mail box. The info is:

```
// Frame buffer offsets

.equ  VID_framebuffer_pwidth,    0x00000000    // Physical Width
.equ  VID_framebuffer_pHeight,   0x00000004    // Physical Height
.equ  VID_framebuffer_vwidth,    0x00000008    // Virtual Width
.equ  VID_framebuffer_vHeight,   0x0000000C    // Virtual Height
.equ  VID_framebuffer_GPU_pitch, 0x00000010    // GPU Pitch
.equ  VID_framebuffer_bitDepth,  0x00000014    // Bit (color) Depth
.equ  VID_framebuffer_x,         0x00000018    // X offset of ULHC
.equ  VID_framebuffer_y,         0x0000001C    // Y offset of ULHC
.equ  VID_framebuffer_GPU_ptr,   0x00000020    // GPU pointer
.equ  VID_framebuffer_GPU_size,  0x00000024    // GPU offset

// Note: GPU pointer and GPU offset are supplied by the Graphics Processor
```

The memory required for a graphics context can be estimated by the number of bytes required by the color depth and the size of the frame area using the following formula:

Bytes_Required = vWidth * vHeight * colorsize, where colorsize = 1 byte for grayscale, 1 byte = low color, 2 bytes for high color, 3 bytes for true color.

4.6 Programming Support

The programming support for the rEOS VMR is a two-pass assembler, which accepts mnemonics matching the rEOS VMR wordcodes and generates matching GNU/Arm source code that is then compiled as part of the customizer build process into the kernel.img file of the working rEOS Operating System, for resident processes, or to a file for non-resident processes.

Like most two-pass assemblers, the rEOS assembler's pass one serves to create a symbol table—used in pass two to resolve label values—and to pre-process the mnemonics—in this case into a java String[] that holds the unlabeled source as an input into pass two.

Pass two of the rEOS JVM assembler processes the symbol table and unlabeled source to create an GNU/Arm .s source files that contains the translated mnemonics, resolved operands, an initialized stack space with stack and static variables, and a predefined PCB with status initialized to new, but not yet scheduled.

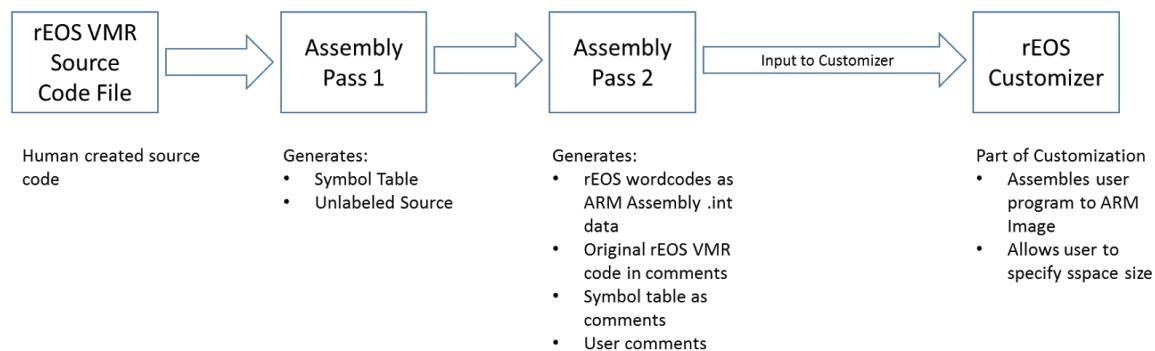


Figure 4.13 User Program Development Overview

4.8 Installation and Customization

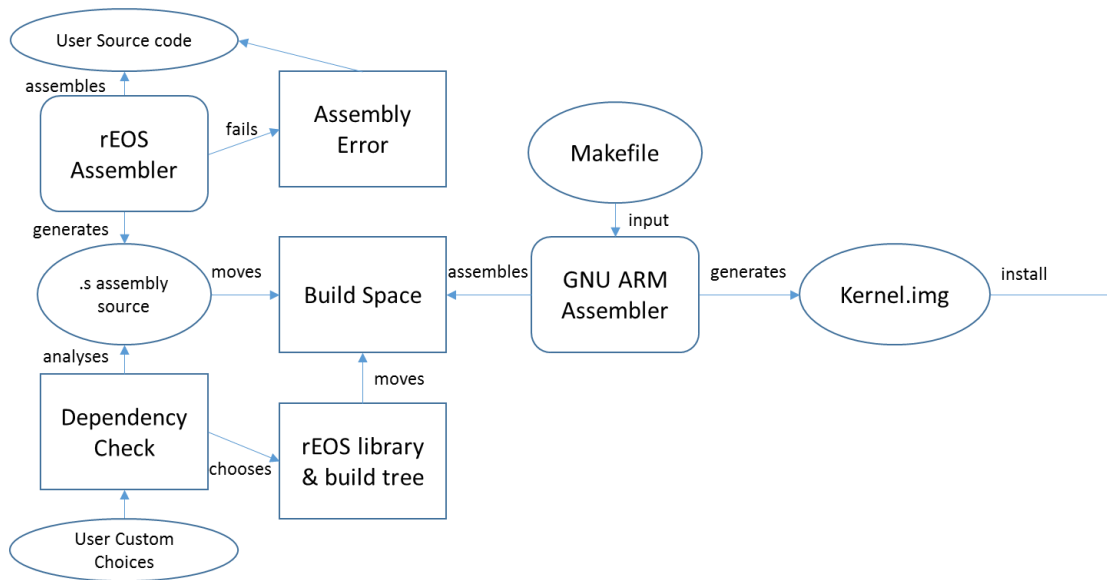


Figure 4.14 Build Process Overview

The installation and customization process allows users to define the hardware configuration of the system being developed, the operating system features required, and the user-written programs to run, and then to build a system image. The process consists of three main steps: configuration, build and install. The configuration and build steps are driven by a Java application dashboard (shown below), and the install step is a simple I/T process.

4.8.1 The Dashboard

The rEOS JVM Assembler, Customization and Build Dashboard controls the configuration and build process.

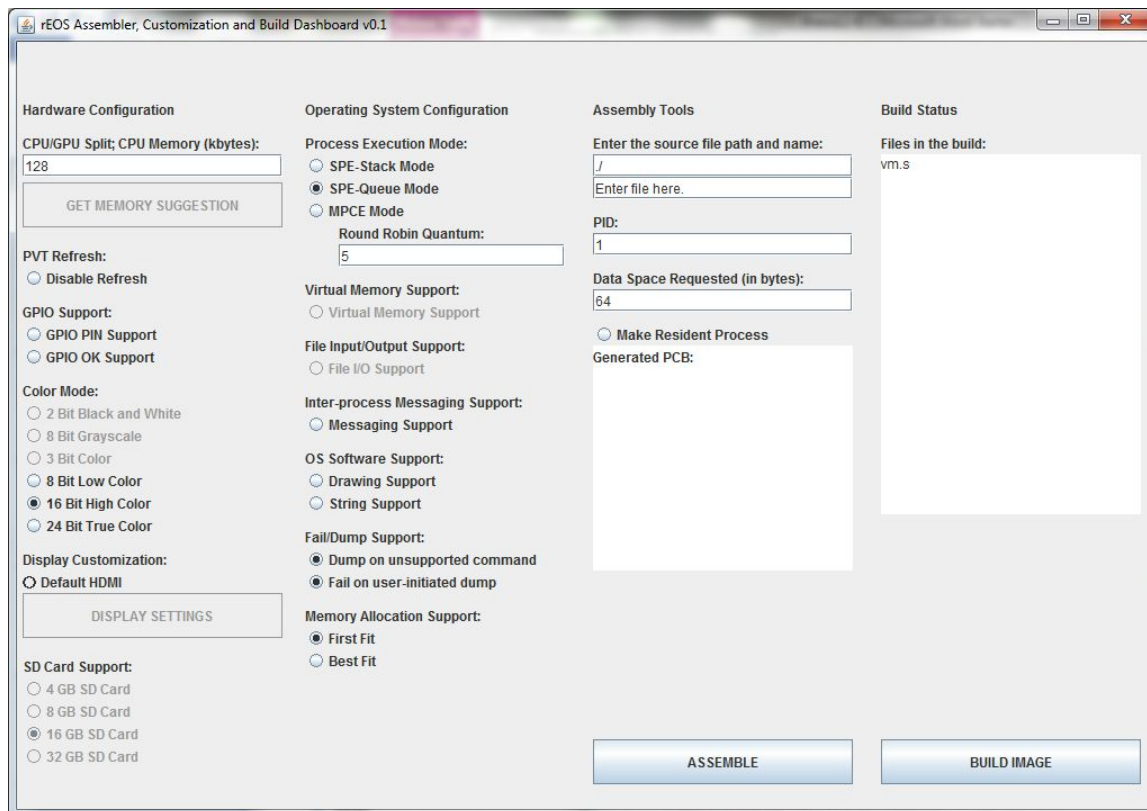


Figure 4.15 Customization Dashboard

The dashboard divides the customization and image-build process into four main sections: Hardware Configuration, Operating System Configuration, Assembly Tools, and Build Status. The first two sections comprise the configuration step and the latter two the build step.

4.8.1.2 Customization

The dashboard organizes customization into two parts: hardware configuration and operating configuration.

4.8.1.2.1 Hardware Configuration

The hardware configuration step allows the user to specify and describe the hardware that the build needs to support: Memory allocation, GPIO support, Color mode, Display mode and SD Card are each separately defined:

- Memory Split between CPU and GPU:
 - The user specifies the size of RPI memory to reserve for the CPU. The customizer uses this parameter to specify config.txt. Note: the customizer will ensure a minimum configuration for CPU size in a future release.
- GPIO Support:
 - The customizer provides a toggle button that allows the user to specify that GPIO support (module gpiopins.s) be included in the build.
 - The customizer provides a toggle button that allows the user to specify that GPIO OK LED support (module gpio.s) be included in the build.
- Color Mode:
 - Users select one of six color modes: 8 bit low color, 16 bit high color, or 24 bit true color. The customizer uses this parameter to set the color mode during initialization.
 - Color mode is ignored if the display setting is No Display.
- Display Customization:
 - A button that launches a dialog that allows the user to select the display for the system or none at all. Because of the large number of Video options for the

Raspberry PI, a separate dialog is needed in the next release. The customizer uses this parameter to set the display more during initialization.

- SD Card Support:
 - The user can specify the size of the SD Card (4 GB, 8 GB, 16 GB, 32 GB). The rEOS uses this parameter statically to manage disk usage instead of dynamically determining the size of the disk (too decrease the image size).

4.8.1.2.1 Operating System Configuration

The Operating System configuration step allows the user to specify and describe what the rEOS build needs to support: process execution mode, virtual memory support, file input/output, inter-process messaging support, drawing or graphics support, fail/dump response, and memory allocation method are each separately defined:

- process execution mode:
 - Three radio buttons allow the user to choose one of SPE Mode (stack enqueue), of SPE Mode (queue enqueue) or MPC mode. This information is used to set internal rEOS variables (schedMode and schedQueueType) which are used to determine which process schedulers to use and how to manage the queue. Note: the Round-Robin quantum is set when choosing the MPCE mode.

Mode Chosen	schedMode value	schedQueueType
SPE-Stack Mode	SCH_MODE_sse (0)	SCH_STATE_stack (1)
SPE-Queue Mode	SCH_STATE_sse(0)	SCH_STATE_queue (0)
MPCE	SCH_STATE_mpce (1)	Value ignored.

Figure 4.16 Process Mode Configuration

- inter-process messaging support:
 - The customizer provides a toggle button that allows the user to specify that inter-process messaging support (module msg.s) be included in the build.
- drawing or graphics support:
 - The customizer provides a toggle button that allows the user to specify that graphics support (module coregraphics.s) be included in the build.
 - The customizer provides a toggle button that allows the user to specify that string support (module string.s, and font.bin) be included in the build.
- fail/dump response:
 - The customizer provides a toggle button that allows the user to specify if an unsupported rEOS VMR instruction should cause a process to abend or continue. This information is used to set an internal rEOS variable (dumpOnUnsupported), which is used by the rEOS VMR.
 - The customizer provides a toggle button that allows the user to specify if a used initiated dump should cause a process to abend or continue. This information is

used to set an internal rEOS variable (dumpOnUserAbend) ,which is used by the rEOS VMR.

- and memory allocation:
 - The customizer provides a pair of radio buttons that allows the user to specify if new processes should be allocated in the first available space in memory or in the smallest available space that will accommodate the new process. This information is used to set an internal rEOS variable (memoryFit), which is used by the rEOS VMR. This functionality is included in this design and the customizer for full implementation in a follow-on project. This functionality will be added when the fork() support is added to the rEOS VMR

4.8.1.2 Build

The dashboard organizes build into two parts: Assembly Tools, and Build Status.

4.8.1.2.1 Assembly Tools

The assembly tools allow the user to assemble programs written in rEOS VMR wordcode mnemonics as either a resident process or as a process that will be forked later. The general process is:

1. Enter the path to the source code file.
2. Enter the source code filename.
3. Specify the sspace required in bytes.
4. Select or not if the process will be resident (this functionality does not exist in this release).

5. Press the Assemble button.
 - a. If the assembly is successful the dashboard status changes to “Success.”
The module is added to the files in build list.
 - b. If the assembly fails the dashboard status is set to a message describing the failure.

4.8.1.2.1 Build Status

The build status displays the files in build list when the user presses Build Image, and it kicks off the execution of the makefile that builds the image. The image and other files can then be transferred to a SD card and placed in the RPi.

4.8.2 Customization Processing

Certain files are included in every build:

- main.s: included because it contains the code to start the rEOS initialization and invocation of other modules.
- processmgr.s: manages the process lifecycle, and is therefore required—even if only one process is running.
- util.s: provides functionality often used in other modules.
- vm.s: controls operation of the rEOS VMR and the execution of many rEOS vm instructions.

The customization dashboard creates the input into the assembly and build step up, which in turn creates the installable files in the image. Specifically:

- config.txt: contains hardware and firmware customizations:

- `init.s`: contains ARM instruction code that configures key rEOS settings during startup.
- source files: the various `.s` source code to include in the build.

The table below summarizes the customization functions.

Option Selected	Files Generated (settings)	Files included
Default Settings	config.txt, init.s	config.txt, init.s, main.s, processmgr.s, util.s, vm.s
CPU/GPU Memory Split	config.txt: sets gpu_mem	config.txt (always included)
PVT Refresh Toggle	config.txt: sets disable_pvt	config.txt (always included)
GPIO Pin Support Toggle		gpiopin.s
GPIO OK Support Toggle		gpiookled
Color Mode Radio Buttons	init.s: sets framebuffer_bitDepth	
Display Dialog	config.txt: sets framebuffer_depth	config.txt (always included)
SD Card Support	tbd	tbd
Process Execution Mode	init.s: <ul style="list-style-type: none"> • Invoke schedulerSetMode • Invoke schedulerDS 	init.s (always included) mpce.s
Virtual Memory Toggle		virtmem.s (Future Release)
File I/O Toggle		fileio.s (Future Release)
Messaging Toggle		processmsg.s
Drawing Toggle		coregraphics.s vid.s
String Toggle		coregraphics.s string.s vid.s
Dump on unsupported command Toggle	init.s: set dumpOnUnsupported	init.s (always included)
Fall on user-initiated dump Toggle	init.s: set dumpOnUnsupported	init.s (always included)
Memory Allocation Model Radio buttons	init.s: set VIRTMEM_first_fit	init.s (always included)
Source Path text field	Used in Assembly	<pgm>.s
Source Name text field	Used in Assembly	<pgm>.s

Make Resident Process	<ul style="list-style-type: none"> ● Used in Assembly, ● configures PCB, ● init.s: set scheduled resident programs. 	init.s
------------------------------	--	--------

Figure 4.17 Customization Relationships

5.0 The Analysis and Experiments

Several phases comprise the rEOS effort: the overall design, the customization design, the proof of concept, and the analysis of the effort and recommendation for follow-up work. Chapter 6 describes the last. The others are explored through the analysis and experiments in this chapter.

5.1 Experiment 1: OS Footprint—Using ARM Assembly Language

One key goal of overall design of the project is to determine if a small footprint, viable operating system. A step in this process was to code the project in ARM assembly.

5.1.1 Description and Rationale

In support of this goal, many of the core features of the rEOS were coded in ARM Assembly (The Broadcom SOC CPU is an ARM Processor), while others are designed for use in a future release. Coding directly to the ARM allows not only the general advantages gained by compressing the data structures and control instructions of a high-level language. It also allowed the use of the ARM registers to pass parameters instead of using the stack. Thus, memory space is saved because a large stack is not required to pass parameters, and storage to memory (and memory to store) moves are limited, thus speeding execution.

5.1.2 Evaluation

The partial memory map from a typical test run below demonstrates the general efficiency of the ARM assembly code. For example, the graphics code that supports creating the graphics context and mailbox (vid.o) requires only 92 bytes of memory, and pixel, line and area fill code (coregraphics.o) requires only 456 bytes of memory. The multi-processing support code (mpce.o and processmgr.o) is only 404 bytes. Examination of the map will reveal other examples.

```

.text          0x00008004      0x15a4
*(.text)
.text          0x00008004      0x1dc  build/coregraphics.o
              0x00008004          graphicSetFGColor
              0x0000801c          setGraphicsADDR
              0x00008028          drawPixel
              0x00008070          drawLine
              0x00008180          fillArea
.text          0x000081e0      0x68  build/font.o
              0x000081e0          drawCharacter
.text          0x00008248      0x40  build/gpioPin.o
              0x00008248          setGPIOPinFunc
              0x00008250          setGPIOPinOn
              0x00008260          setGPIOPinOff
              0x00008270          GPIOPinRead
.text          0x00008288      0x78  build/gpiokled.o
              0x00008288          getGPIOAddress
              0x00008290          setGPIOFunction
              0x000082c4          setGPIO
.text          0x00008300      0x13c build/init.o
              0x00008300          initREOS
.text          0x0000843c      0x54  build/main.o
              0x00008444          _end
.text          0x00008490      0x2f0 build/mpce.o
              0x00008490          schedulerMPCEAdd
              0x00008518          schedulerMPCERun
.text          0x00008780          0x0  build/ostester.o
.text          0x00008780      0x194 build/processmgr.o

```

	0x00008780		setQuantum
	0x0000878c		schedulerSetMode
	0x000087a4		schedulerSetDS
	0x000087bc		schedulerSPE
	0x000087e8		
schedulerSPEAddStack			
	0x00008820		
schedulerSPEAddQueue			
	0x00008854		schedulerSPERun
.text	0x00008914	0x1cc	build/processmsg.o
	0x00008914		msgOpenChk
	0x00008978		msgOpenChkLock
	0x0000898c		msgOpenLock
	0x000089a0		msgOpenDone
	0x000089bc		msgWrite
	0x00008a4c		msgRead
.text	0x00008ae0	0x0	build/reader.o
.text	0x00008ae0	0x48	build/string.o
	0x00008ae0		drawString
.text	0x00008b28	0x3c	build/time.o
	0x00008b28		getTimerBase
	0x00008b30		getTimeStamp
	0x00008b3c		waitForTimer
.text	0x00008b64	0x0	build/timeTestPgm01.o
.text	0x00008b64	0x50	build/util.o
	0x00008b64		clearMem
	0x00008b80		div
	0x00008ba0		mod
.text	0x00008bb4	0xc8	build/vid.o
	0x00008bb4		vidMailBoxWrite
	0x00008be0		vidMailBoxRead
	0x00008c10		vidSetFrameBuffer
.text	0x00008c7c	0x0	build/virtmem.o
.text	0x00008c7c	0x92c	build/vm.o
	0x00008c7c		VM_main
.text	0x000095a8	0x0	build/writer.o

In subsequent releases, more space could be saved by allowing the customization of the stack size. Currently, the stack size is statically set in main.o to 0x8000 bytes. This value could

be evaluated by the builder and adjusted downward to provide a more efficient use of memory by adding a new customization option to define stack size.

5.2 Experiment 2: Customization--Footprint

One key goal of this project is to determine if a small footprint, viable operating system. In support of this goal, many of the core features of the rEOS were created. But all components of the OS do not need to be installed to create a workable program. The user/builder can pick and choose the features required for their environment, and thus control the size of the installable components.

5.2.1 Description and Rationale

The table in 4.8.3 outlines the files (and settings) driven by customization. The various configurations result in different image sizes. The table below shows the memory when a configuration option is chosen. By only choosing the functionality needed, the footprint can be reduced. Modules in italics are included in every build.

Option Selected	Source File	.text Memory size (bytes)	.data Memory Size (bytes)
Default (no options)	<i>config.txt</i>	n/a	n/a
	<i>init.s</i>	316	0
	<i>main.s</i>	84	0
	<i>processmgr.s</i>	460	20
	<i>util.s</i>	80	0
	<i>vm.s</i>	2348	100
GPIO Pin Support	gpio.s	120	0
GPIO OK Support	gpiookled.s	64	0
MPCE Mode	mpce.s	752	0
Virtual Memory	virtmem.s	future release	future release
File I/O	fileio.s	future release	future release
Messaging	processmsg.s	460	20
Drawing	coregraphics.s	476	8
	vid.s	200	4096
String Support	coregraphics.s	476	8
	string.s	74	0
	vid.s	200	4096

Figure 5.1 Example Build Configuration

5.2.3 Evaluation

The modular nature of the rEOS allows the builder to control the installable build size. A minimum build with the core modules and GPIO Pin support requires 3528 bytes of resident executable memory, while even the maximum build is less than 21,000 bytes.

5.3 Experiment 3: Processes and Memory Segmentation

To further the small footprint goal of the rEOS, the fork() wordcode design leverages the natural memory segmentation of rEOS software program's separate pspace (text, executable wordcodes), and sspace (data). Note: Storage resources are not explored or managed in this release, but should be addressed in future work.

5.3.1 Description and Rationale

The fork() design is intended to be implemented as a follow-on project. The plan is to implement a traditional segmentation scheme, with a minor twist. By customization design, all rEOS programs load into memory as resident programs. Therefore, all programs running or dormant (in memory, but not running) are resident in memory. Since all running processes of the same program share the same pspace, it is easy enough to take advantage of segmentation by allowing multiple versions to run by assigning that pspace to new forked versions of the process, while assigning a new, separate sspace to the new process.

5.3.2 Evaluation

Realistically, the footprint savings of this method is limited to that of any segmentation memory scheme. Savings are only realized when more than one copy of a process is needed, and at least one copy of each process is running.

Consider that if a process runs and exits, the rEOS does not remove the process from memory and return that space to the available pool. Removing the process pspace would require that the process either a) cannot be run again before a reboot, or b) must be reloaded from

storage before being restarted. Given these possibilities, the rEOS processes remain in memory. The repercussion is that pspace memory occupied by a process is wasted, unless it is still active.

If only one process copy is running, that process has a pspace and sspace. They represent the basic memory requirement for a running process. Segmentation in the rEOS only represents a savings when more than one copy of the process runs, resulting in one pspace and 2 or more sspaces.

5.4 Experiment 4: Atomic Message Execution

One key goal of this thesis is to leverage the run-time environment to provide complex functions (i.e. inter process messaging and GPIO support) that run atomically (non-interruptible).

5.4.1 Description and Rationale

Multi-processing in a native hardware environment problematizes the programs that require that multiple machine instructions run uninterrupted. This is the well-known Critical Section Problem. The rEOS VM runs each wordcode as one atomic instruction because it measures quantum of execution by wordcodes rather than clock cycles. Thus, regardless of the number of machine level instructions that comprise the wordcode, all of the instructions that make up each wordcode run atomically.

The rEOS support two types of complex wordcodes: GPIO support and interprocess messaging. Both key wordcodes meet the core requirements of a critical section problem.

5.4.2 Evaluation

GPIO support reaps two advantages. First, it allows an abstraction of the interface to the GPIO support described in the *BOARDCOM BCM2835 ARM Peripherals* document, allowing

the user to take advantage of using the hardware support, while providing an easy to use interface. Second, each read or write to the GPIO requires multiple machine level instructions and the GPIO support insures that they run as critical sections.

This solution ensures mutual exclusion by making the wordcode atomic. It ensures that progress is made because each instruction is bounded in execution time in length, and all processes will eventually run because the mpce is a round-robin queue.

6.0 Conclusion

The goals of the original project were met. Ideally, the messaging code and the memory management options code would have been completed to allow for more benchmarking.

However, the design progressed sufficiently to code significant portions of the rEOS to suffice as a proof of concept. Moreover, several key ideas crystallized during the project as possible follow-ons.

6.1 Observations and Lessons Learned

Beyond the specific realm of findings related to the thesis, several other lessons can be learned from the work done for this thesis. They are enumerated here in no particular order.

6.1.1 Small Footprint

As shown in the experiments in the previous section, it is clear that a very small footprint image is possible note that the Raspbian image requires about 33 MB as compared to the 20KB requirement of the rEOS. Of course, the Raspbian might be paired down by removing modules that are not needed by the task at hand. But most of the saving would be storage savings, not memory savings.

6.1.2 Complex Atomic Instructions

Perhaps the most interesting aspect of the work was the creation of the complex, wordcodes. A follow-on effort might explore this idea in greater detail, creating a virtual machine dedicated to a narrow range of problems such as neural networking or artificial life applications.

Since the widespread use of the general purpose computer and general purpose operating system, such specialized systems have not been broadly used. But the apparent slow-down of Moore's law, and the low cost of the Raspberry Pi and similar computers on a card make this area worthy of further study.

6.1.3 Software Engineering

As the project progressed, I added to the scope of the work. As a former software development project manager at a global technology corporation, I am especially conscious of the problem of scope creep. Nevertheless, I found that my expectations of a modern operating system caused me to enlarge the project beyond the original scope.

This problem might have been prevented by applying a more heavy-weight software engineering process, such as a modified waterfall, instead of the lightweight iterative process used. The required high-level of interaction and inter-dependencies required by an operating systems would have better addressed this way.

6.2 Final Thoughts

This project explored many aspects of Operating Systems. It identifies opportunities for future investigation, in particular the Internet of Things and additional hardware support. It validated the viability of a micro-operating system for an EOS-like system, and uncovered several areas for future study.

7.0 Appendix II, Abbreviations and Glossary

C

CoAP: Constrained Application Protocol. A specialized, lightweight web transfer protocol often leveraged by the IoT

E

Embedded Device: A special purpose computing device. An example is the computer chip used to monitor the status of systems in an automobile.

Embedded Operating System: An efficient, small memory and storage footprint operating system customized for a specific purpose at installation time that abstracts hardware interaction through a specifically defined set of operating system and software support.

EOS: Embedded Operating System.

I

IETF: the Internet Engineering Task Force. An open-source community with the goal to “make the Internet work better.”

IoT: the Internet of Things. An approach to the creation of devices that look like ordinary things, but leverage an Internet connection to provide greater functionality. An example is an umbrella that connects to the Internet to determine the weather forecast and blinks a light when rain is expected.

P

PCB: Process Control Block

R

RTOS: Real-time Operating System

S

SOC: System on a Chip

8.0 Overview of the rEOS Virtual Machine Runtime

The rEOS Stack Machine Runtime (rEOS VMR) is the execution environment for user programs in the rEOS. Its design is based on my previous project called the General Purpose Virtual Machine (GPVM). The GPVM is intended to solve basic algebra problems, using integer arithmetic. Its purpose is to provide a robust platform that allows students of computer science to learn the basics of virtual machine internals by extending its provided opcodes or by designing their own set of opcodes. The GPVM is written in Java and can be extended in that environment.

The rEOS VMR adapts some of the underlying principles of the GPVM. These include:

- Two hidden registers: the usual program counter (pc) and stack pointer (sp). The GPVM's parameter start register (ps) is not needed in the rEOS VMR.
- Separate read-only program and read-write stack spaces, to eliminate possible runtime corruption.

Yet the rEOS VMR is an entirely different project because:

- It is written in Arm Assembler instead of Java.
- It is written to bare metal, and not to an existing VM.
- It provides specific access to RPi hardware and rEOS support.
- It supports a wordcode compiler.

8.1 rEOS VMR Design Overview

The rEOS VMR has several main features discussed in the following subsections

8.1.1 Virtual Registers and Memory

The rEOS VMR is a virtual_machine (VM) based on a stack architecture. Before looking at the core code, one needs an understanding of the rEOS VMR's use of physical and virtual registers, and program and stack spaces.

The rEOS VMR runs in the User ARM operating mode and thus has access to registers r0-r15 (as well as the User CPSR). The ARM Application Binary Interface (ABI) requires that the contents of registers r4-r12 contain the same values on return from a linkage call as when the linkage call occurred. The ABI defines registers r0-r3 as true "scratch registers, usable by the rEOS VMR as needed, while r4-r12 contents must be pushed to the stack upon entry to the rEOS VMR and restored upon exit.

The working rEOS VMR must use registers for memory independent operations in order to optimize execution time. Thus the variable data needed for memory protection and instruction decoding are stored in registers r4-r12. Generally, these content registers do not need to be stored on the stack, which saves expensive register to memory or memory to register moves. The lpc (r4) is an exception to this rule, because the drawstring instruction requires an additional data space. The limited number of free registers makes it impossible to contain all the needed data within them, and therefore some data needs to be kept outside the registers in memory. To limit the performance impact of this, the rEOS VM keeps the number of cycles to execute within the PCB_cycles field. Since it is accessed only once per wordcode execution. This has the effect of evenly spreading the cost of the memory to register accesses evenly across all instruction execution.

The rEOS VMR uses dedicated internal registers shown in the following table:

rEOS VMR Register Name	Physical Register	Purpose	Stacked?
lpc	r4	Logical Program Counter	Yes
lsp	r12	Logical Stack Point	No
psl	r5	Program Space Limit Register	No
spl	r6	Stack Space Limit Register	No
pspace	r7	Program Space Relocation Register	No
sspace	r8	Stack Space Relocation Register	No
vmpc	r9	VM Physical PC Register	No
vmsp	r10	VM Physical SP Register	No
vmdc	r11	VM Decoding Register	No
PBCAddr	n/a	Address of the PBC	Always

Figure 8.1 Internal rEOS VMR Mapping

The alternative was to save the data in a scratch register, and push it to the stack whenever that register was needed or whenever a function call was made. While this would preserve the speed of many common instructions, the slowest instructions would be impacted the most, because they make the most use of scratch registers and make the most function calls.

When the rEOS Process Scheduler is invoked, it allocates memory to hold the user program and program stack space based on the program's profile, if required. Next, based on the program profile, the input data and rEOS calculations, the rEOS VMR allocates a memory space—called `stackSpace`—to the program, with any input data to process and enough extra space to support computations by the VM. At the end of execution, the `stackSpace` contains any output from the VM.

In many stack architectures, it is common to place the stack in the same array as the program in some VM designs. This can result in the unintentional (or intentional) overwriting of the executable code during stack operations. The rEOS VMR's separate stack and program

arrays prevent this problem because the rEOS VMR does not supply any method to allow user program alteration of the program space during execution. It also allows the rEOS to exploit segmentation.

The registers are described in the table above in general terms, but more details on the specifics can be found elsewhere in this thesis.

8.1.2 rEOS VMR Wordcodes

User programs are written using a text file and custom assembler and results in bytecodes for the rEOS VMR. It should be noted that all wordcodes and data objects are 32 bit words; increment and decrement operators should be understood as updates to the pointer to the next (or preceding) word not byte address. The methods provided are enumerated here with their c-like pseudo-code to describe their function:

Instruction	c-like pseudo-code	Description
add()	stack[sp++] = stack[sp]+stack[sp+1]	pop the top two elements on the stack and push their sum.
and()	stack[sp++] = stack[sp] BITWISE AND stack[sp+1]	pop the top two elements on the stack and push result of bitwise anding them.
asl()	stack[sp++] = stack[sp+1] << stack[sp]	Arithmetic shift left
asr()	stack[sp++] = stack[sp+1] >>>stack[sp]	Arithmetic shift right.
bin2str()	stack[sp] = (String)stack[sp]	Convert int value to ASCII String for display.
call()	stack[sp] = pc+2 pc = pspace[pc+1]	branch with linkage
dec()	stack[sp]--	decrement the value on top of the stack.
div()	stack[sp++] = stack[sp+1] /stack[sp]	Integer division
drawcharacter(c,x,y,n)	fb[x,y]..fb[x+8,y+16] = n	Draw ascii character n with its ULLC at (x,y) and LRHC at

		(x+8, y+16) using the installed font in the color c.
drawline(x0,y0,x1,y1,c)	fb[x0,y0]:fb[x1,y1]=c	Uses Bresenham's Algorithm to compute the pixels to draw a line from (x0,y0) to (x1, y1) the color c.
drawpixel(c,x,y)	fb[x,y]=c	set the pixel on the current screen at position (x, y) to the color c
drawstring(n,s)	fb[x,y].. fb[x+(8*N), y+16]=S	Iteratively calls drawcharacter()to draw n characters starting at position (x,y).
dup()	stack[--sp]= stack[sp+1]	Duplicate the top value on the stack.
dump()	(Scheduler Request)	pause the execution of the program and create a file containing the contents of the registers, the program space and the stack space. and exit the rEOS VMR
fork()	(Future Release)	fork a new process through the creation of a new PCB and place in the new state.
fill(x,y,w, h,c)	fb[x,y]:fb[x+w,y+h]=c	Fills a rectangle with the color c.
fcloser()	filesystem request	Close the file opened for reading.
fclosew()	filesystem request	Close the file opened for writing.
fopenr(n)	filesystem request	Open a file named n for reading.
fopenw(n)	filesystem request	Open a file named n for writing
fread()	stack[--sp]= word	reads one word from open read file and pushes it to the stack.
fwrite()	word = stack[sp]++	Writes top value of stack to the open write file..
getgraphics()	stack[sp--]= mem[graphicsBuffer]	Creates a framebuffer to draw graphics to. [Restored to thesis to allow for alternate output screen sizes.]
gpioread()		Provides a read of a GPIO Address

gpiowrite()		Provides a write to a GPIO Address
inc()	stack[sp]++	increment value atop the stack
jmp()	pc= pspace[pc+1]	unconditional jump
jmpeq()	stack[sp]-stack[sp+1]; if (z==1) pc= pspace[pc+1]; else pc++;	Jump to n if top two stack values are equal.
jmpgt()	stack[sp]-stack[sp+1]; if (n==0&&z==0) pc= pspace[pc+1]; else pc++;	Jump to n if top stack value is greater than the value under it.
lsr()	stack[sp]>> stack[sp++]	Shift right logical.
msgwrite()	if (stack[0]==0 (unwritten){ stack[2..n+2] = data stack[1] = length of data stack[0] = 1 (written) }	Writes data to the message buffer.
msgread()	if (stack[0] ==1 (written){ stack[vmsp..vmsp+stack[1] = stack[2..n+2] ; stack[0] = 2 (nullbuffer) }	Read data from the message buffer.
msgOpen(n)	if (sender.stack[0]== 0 (nullbuffer){ stack[1] = n+2 (buff size); stack[0] = 1 unwritten; }	creates a msg buffer within the sspace of the sending program. This buffer begins at the lowest stack address and grows towards the stack. Buffers may be written to once and read from once. A new buffer can only be made if stack[0]=2 (read).
mod()	stack[sp++] = stack[sp+1] mod stack[sp]	Modulus
mul()	stack[sp++] = stack[sp]*stack[sp+1]	pop the top two elements on the stack and push their product.
neg()	stack[sp]= -stack[sp]	Negate the top value in the stack
nop()		No operation.
not()	stack[sp]= BITWISE NOT stack[sp]	pop top of stack and push its bitwise inverse.

or()	stack[sp++]= stack[sp] BITWISE OR stack[sp+1]	pop the top two elements on the stack and push result of bitwise oring them.
pause()	(Scheduler Request)	return remaining time slice to the rEOS VMR, without specifying conditions for restart.
pausew()	(Scheduler Request) stack[sp++]	return remaining time slice to the rEOS VMR, with request to restart after stack[sp++] milliseconds. Note: this commands only guarantees the scheduler will not restart the process before that time, but no guarantees are made about how long after.
push(n)	stack[--sp] = n	immediate push of a 32-bit data word onto the stack.
pushi()	stack[--sp] = sp[space- stack[sp++]]	push the data from the specified position in the stack to the top of the stack.
pushsp()	stack[--sp] = sp	push the current value of the sp to the top of the stack.
pop()	sp++	move the stack pointer.
popsp()	sp = stack[sp++]	Pop the top value in the stack into the sp register
ret()	pc=stack[sp++]	Linkage return
ror()	stack[sp++]= rollright(stack[sp++])	rotate the word right
sub()	stack[sp++] = stack[sp+1]-stack[sp]	Subtract the top of stack from the next value.
swap()	temp=stack[sp] stack[sp]=stack[sp+1] stack[sp+1] = temp	Swap the top two values on the stack.
time()	stack[--sp] = mem[0X2000300A]	Pushes the low order word of the system time to the stack.
xor()	stack[sp++]= stack[sp] BITWISE OR stack[sp+1]	pop the top two elements on the stack and push result of bitwise xoring them.

Figure 8.1 Wordcode Table

9.0 Source Code

There are four separate groupings of source code for the rEOS:

- The first is the ARM Assembly source that makes up the installable components of the operating system, shown here in section 9.1.
- The second is the Java Customizer code included in section 9.2.
- The third is the Java Customizer generated files. Section 9.3 holds samples of this generated text.
- Lastly, sample rEOS VMR code, used to test during development.

9.1 rEOS ARM Assembly Source Code

This section contains the bulk of source code for the rEOS.

9.1.1 coregraphics.s

```
//-----
// coregraphics.s:  Base Graphics support for rEOS VMR
//
// Copyright (c) 2016, 2015, 2014 by James J. Perry
//-----
// Equates
//-----
.equ      GRA_max_u16_color,          0x010000

//-----
// Graphics Data
//-----
.section .data

.align 1
fgColor:      .hword 0xFFFF      // Foreground Color
.align 2
graphics_ADDR: .int 0            // & Frame Buffer Info
```

```

.section .text
//-----
//- graphicSetFG: Sets the foreground color.
//-
//- Input Registers:
//-      r0:  a U16 color to set as the current color to write.
//-
//- Return Registers:
//-      none
//-----
.globl graphicSetFGColor
graphicSetFGColor:

color          .req r0          // new foreground color
fColor         .req r1          // Address of the foreground
                                   // color data area.

cmp            color, #GRA_max_u16_color
movhi         pc, lr           // bad color value
moveq         pc, lr           // bad color value
ldr           fColor, =fgColor // save the color
strh          color, [fColor]
mov           pc, lr
.unreq        color
.unreq        fColor
//-----
//-setGraphicsADDR
//-
//- Sets the Frame buffer addr
//-
//- Parameters:
//-      r0:  addr of the new frame buffer
//-
//- Return Registers:
//-      none
//-----
.globl setGraphicsADDR
setGraphicsADDR:

bufferAddr    .req r0          // address of the Info Buffer
graphicAddr   .req r1          // Graphic addr
ldr           graphicAddr, =graphics_ADDR
str           bufferAddr, [graphicAddr]
mov           pc, lr
.unreq        bufferAddr

```

```

.unreq          graphicAddr
//-----
// - drawPixel
// -
// - Writes a pixel of the current color to the current
// - buffer. Fails silently on bad x or y
// -
// - Parameters:
// -     r0:  x coordinate
// -     r1:  y coordinate
// -
// - Return Registers:
// -     none
//-----
.globl drawPixel
drawPixel:

x                .req r0                // x-coord
y                .req r1                // y-coord
addr             .req r2                // address for write
ldr              addr, =graphics_ADDR
ldr              addr, [addr]
//-----
// - Verify Position
//-----
max              .req r3

ldr              max, [addr, #4]
sub              max, #1
cmp              y, max
movhi            pc,lr

ldr              max, [addr, #0]        // x is checked second, to
// be sure the value is in
// max for next step.

sub              max, #1
cmp              x, max
movhi            pc,lr

ldr              addr, [addr, #32]
add              max, #1
mla              x, y, max, x
add              addr, x, lsl #1

```

```

color          .req r3
ldr            color, =fgColor
ldrh          color, [color]
strh          color, [addr]

mov            pc, lr
.unreq        x
.unreq        y
.unreq        addr
.unreq        max

//-----
// - drawLine
// -
// - Writes a line of the current color to the current buffer.
// - Draws line of pixels between points. Fails silently on
// - any pixel with bad x or y.
// -
// - Parameters:
// -     r0:  x0 coordinate
// -     r1:  y0 coordinate
// -     r2:  x1 coordinate
// -     r3:  y1 coordinate
// -
// - Return Registers:
// -     none
//-----

.globl drawLine
drawLine:
x0            .req r0
y0            .req r1
x1            .req r2
y1            .req r3

        // Save lr--since we will always attempt a call.
push        {lr}
        // if x1 is right of x0  then go to skip swap draw
cmp         x0, x1
blt         skip
bgt         skip

        // if vert and first above second go to skip swap draw
cmp         y0, y1
blt         skip

```

```

        // x0 right of x1, swap left & right, goto skip swap draw
push     {x0,y0,x1,y1}
pop      {x1,y1}
pop      {x0,y0}

skip:
x        .req r0          // x of point to draw
y        .req r1          // y of point to draw
endX     .req r4          // x of endpoint (x1)
endY     .req r5          // y of endpoint (y1)
nextX    .req r6          // x for next draw
nextY    .req r7          // y for next draw
push     {r4, r5, r6, r7}
        // set up line draw by pixel by initializing registers
mov      endX, x1
mov      endY, y1
mov      nextX, x0
mov      nextY, y0

        // if not vert go to normal draw
cmp      x, endX
bne     normal
        // else draw first point then loop to end

vLine:
mov      x, nextX
mov      y, nextY
bl      drawPixel
        // loop and define next point until endX endY
add      nextY, #1
cmp      nextY, endY
blt     vLine
beq     vLine
b       done

normal:
        // Use Bresenham's Algorithm to draw lines
dx       .req r8          // delta x
dy       .req r9          // delta y
err      .req r10         // err size:
                        // used to determine
                        // when to use next y
up       .req r11         // 1 if Positive Slope,
                        // else 0

```

```

    // Save registers used for Bresenham's Algorithm
push    { r8, r9, r10, r11}
    // determine deltas, err, inc
mov     dx, x1
sub     dx, x0           // dx = x1-x0
mov     dy, y1
sub     dy, y0           // dy = y1-y0
mov     err, dy
add     err, dy
sub     err, dx           // err = dy*2-dx
mov     up, #0
cmp     y1, y0
movlt   up, #1           // if y1<y0 up = 1; else up
=0;

    // loop through x-ccords until end.
otherLine:
mov     x, nextX
mov     y, nextY
bl     drawPixel
    // compute next point
cmp     up, #1
beq     upElse
cmp     err, #0
beq     else
blt     else
add     nextY, #1         // nextY++
add     err, dy
add     err, dy
sub     err, dx
sub     err, dx           // err += dy*2-dx*2
b     endLoop           // loop back
else:
add     err, dy
add     err, dy
b     endLoop
upElse:
cmp     err, #0
beq     else
bgt     else
sub     nextY, #1         // nextY--
add     err, dy
add     err, dy
sub     err, dx
sub     err, dx           // err += dy*2-dx*2

```

```

b            endLoop
    // loop until next x > endx
endLoop:
add         nextX,#1
cmp         nextX,endX
blt        otherLine
beq        otherLine
    // Restore Bresenham's Algorithm registers
pop        { r8, r9, r10, r11 }

done:
    // restore common registers before return
pop        {r4, r5, r6, r7}
    // return
pop        { pc }

.unreq     x0
.unreq     y0
.unreq     x1
.unreq     y1
.unreq     x
.unreq     y
.unreq     endX
.unreq     endY
.unreq     nextX
.unreq     nextY

//-----
// - fillArea
// -
// - Writes a pixel of the current color to the current
// - buffer. Fails silently on bad pixel x or y.
// -
// - Input Registers:
// -     r0:  x coordinate
// -     r1:  y coordinate
// -     r2:  width
// -     r3:  height
// -
// - Return Registers:
// -     none
//-----
.globl fillArea
fillArea:

```

```

nextX      .req r6          // x-coord
nextY      .req r7          // y-coord
maxX       .req r4
maxY       .req r5
zeroX     .req r8          // x coord left most
push      {r4, r5, r6, r7, r8, lr }
          // Initialize local variable
mov       nextX, r0
mov       nextY, r1
mov       zeroX, r0
          // set LRHC
mov       maxX, #0
mov       maxY, #0
add      maxX, nextX, r2
add      maxY, nextY, r3
          // Start x again on next row. Exit if filled.
GRA_FILL_rowLoop:
cmp      nextY, maxY
bgt      GRA_FILL_exit
mov      nextX, zeroX

          // Fill the column for this row.
GRA_FILL_colLoop:
cmp      maxX, nextX
blt      GRA_FILL_colExit
mov      r0, nextX
mov      r1, nextY
bl       drawPixel
add     nextX, #1
b       GRA_FILL_colLoop

GRA_FILL_colExit:
add     nextY, #1
b       GRA_FILL_rowLoop

          // restore stack and return
GRA_FILL_exit:
pop     {r4, r5, r6, r7, r8, lr }
mov     pc, lr
.unreq  nextX
.unreq  nextY
.unreq  maxX
.unreq  maxY

```

```
.unreq          zeroX
```

9.1.2 font.s

```
//-----
// - Font and draw character support for rEOS VMR
//
// Copyright (c) 2015, 2014 by James J. Perry
//-----
// - Equates
//-----
.equ          FON_ascii_max,          0x0000007F
//-----
// - Char Graphics Data
//-----
.section .data

.align 4
.globl font
font:
.incbin "font.bin"

.section .text
//-----
// - drawCharacter
// -
// - Function to draw a single ASCII character (codes 0x00 to
// - 0xFF). Fails silently on illegal write or unsupported
// - ASCII code.
// -
// - Input Registers:
// - None.
// -
// - Return Registers:
// - r0 : contains the ascii code for the character
// - r1 : contains the ULHC x coord to draw to
// - r2 : contains the ULHC y coord to draw to
// -
// -
//-----
--
.globl drawCharacter
```

drawCharacter:

```

cmp          r0,#FON_ascii_max
movhi       pc,lr          // Exit on bad ASCII
push        {r4,r5,r6,r7,r8,lr} // Registers used for
                                     // vars and lr
nextX       .req r4        // nextX to draw
nextY       .req r5        // nextY to draw
charAddr    .req r6        // addr of bitmap byte

                // copy address to local variable (avoid call
overwrite)
mov         nextX,r1
mov         nextY,r2
                // Get the first position in character by charaddr
                // = (FontStart+ character code*16) each character
                // is 8 x 16 bits or 16 bytes.
ldr        charAddr, =font
add        charAddr, r0, lsl #4

```

FON_lineLoop:

```

bits        .req r7        // holds the line data.
bit         .req r8        // Holds bit to process.
                // Load top 8 bits of the character,
                // and seed bit to be left bit -1;
ldrb       bits,[charAddr]
mov        bit,#8

```

FON_colLoop:

```

                // indicate next bit (while not end of byte)
subs       bit,#1
blt        FON_colLoopEnd // process next row
lsl        bits,#1        // shift to desired bit

                // Write bit if on
tst        bits,#0x100    // test if a bit on
beq        FON_colLoop    // loop back if not
add        r0,nextX,bit    // Compute next position
mov        r1,nextY
bl         drawPixel      // draw
                // determine if end of row loop back if not
teq        bit,#0
bne        FON_colLoop

```

FON_colLoopEnd:

```

                // Set Next line
add             nextY,#1
add             charAddr,#1
tst             charAddr,#0b1111
bne             FON_lineLoop
.unreq bit
.unreq bits
.unreq nextX
.unreq nextY
.unreq charAddr
pop             {r4,r5,r6,r7,r8,lr}
mov             pc, lr

```

9.1.3 gpiokled.s

```

//-----
//- gpiopin.s GPIO support for OK_LED
//
// Copyright (c) 2016, 2015, 2014 by James J. Perry
//-----
//- Equates
//-----
.equ GPIO_ADDR_$gpio, 0x20200000

//-----
//- getGPIOAddress
//-
//- Function to get the start address of the GPIO controls.
//-
//- Input Registers:
//-     None.
//-
//- Return Registers:
//-     r0 : contains the start address of GPIO
//-
//-----
.globl getGPIOAddress

getGPIOAddress:
gpioAddr .req r0
ldr      gpioAddr, =GPIO_ADDR_$gpio
mov      pc, lr

```

```

.unreq    gpioAddr

//-----
// - setGPIOFunction
// -
// - Function to set the GPIO controls.
// -
// - Input Registers:
// -   r0 : GPIO Pin Number
// -   r1 : Function Number
// -
// - Return Registers:
// -   r0 : contains the start address of GPIO
//-----

.globl setGPIOFunction

setGPIOFunction:
pinNum    .req r0
funcNum   .req r1

cmp       pinNum, #53           // Validate Pin Number
cmpls    funcNum, #7           // Validate Function
movhi    pc, lr                // invalid fails silently.
push     {lr}

gpioAddr  .req r2

ldr      gpioAddr, =GPIO_ADDR_$gpio

loop1_setGPIOFunction:
cmp      pinNum, #9             // determine the addr
subhi    pinNum, #10
addhi    gpioAddr, #4
bhi     loop1_setGPIOFunction

add      pinNum, pinNum, lsl #1  // r2 * 3 (3 bits per pin)
lsl     funcNum, pinNum         // shift to 3correct pin
str     funcNum, [gpioAddr]

.unreq   gpioAddr
.unreq   funcNum
.unreq   pinNum
pop     {pc}

```

```

//-----
//- SetGPIO
//-
//- Sets the GPIO function.
//-
//- Input Registers:
//-     r0: GPIO Pin number
//-     r1: Value to set
//-
//- Returns Registers:
//-     none.
//-----

.globl setGPIO
setGPIO:
pinNum .req r0
pinVal .req r1

cmp     pinNum, #53          // ensure pin# is valid
movhi   pc, lr              // if invalid pin silently fail
push    {lr}

gpioAddr .req r2

ldr     gpioAddr, =GPIO_ADDR_$gpio

pinBank .req r3
lsr     pinBank, pinNum, #5 // Determine if GPIO addr is
                          // 1st or 2nd word, int div
                          // by 32
lsl     pinBank, #2         // and multiply by 4
add     gpioAddr, pinBank  // Compute final address.

.unreq  pinBank

and     pinNum, #31         // (effect of n mod 32).

setBit  .req r3

mov     setBit, #1
lsl     setBit, pinNum     // Selects the bit to set.
teq     pinVal, #0         // Is the pinval 0?
streq   setBit, [gpioAddr, #40] // Turn pin off,
                          // if requested

```

```

strne    setBit, [gpioAddr, #28]    // turn pin on if
                                           // requested.

.unreq   pinNum
.unreq   pinVal
.unreq   setBit
.unreq   gpioAddr

pop      {pc}

```

9.1.4 gpiopin.s

```

//-----
//- gpiopin.s GPIO support for GPIO PIN input/output
//
// Copyright (c) 2015, 2014 by James J. Perry
//-----
//- Equates
//-----
//.equ GPIO_ADDR_$gpio, 0x20200000
//-----
//- setGPIOPinFunc
//-
//- Function to set the GPIO controls. No checking is done;
//- instead rEOS requires parameter checking in assembler
//- Parameters:
//-    r0 : GPIO Pin Number address
//-    r1 : Function Number constant
//- Return values:
//-    r0 : contains the start address of GPIO
//-
//-----
.globl setGPIOPinFunc
setGPIOPinFunc:

pinVal    .req r0
funcVal   .req r1

str       funcVal, [pinval]
mov       pc, lr

.unreq    pinVal
.unreq    funcVal

```

```

//-----
//- setGPIOPinOn
//-
//- Sets the GPIO Pin to on
//- Parameters:
//-     r0: GPIO Pin number
//-     r1: Value to set
//-
//- Returns:
//-
//-----
.globl setGPIOPinOn
setGPIOPinOn:
pinAddr .req r0
pinVal .req r1

ldr     r2,[pinAddr]
orr     r2,r1
str     r2, [pinAddr]

.unreq pinAddr
.unreq pinVal
mov     pc, lr
//-----
//- setGPIOPinOff
//-
//- resets the GPIO Pin to off
//- Parameters:
//-     r0: GPIO Pin number
//-     r1: Value to set
//-
//- Returns:
//-
//-----
.globl setGPIOPinOff
setGPIOPinOff:
pinAddr .req r0
pinVal .req r1

ldr     r2,[pinAddr]
and     r2,r1
str     r2, [pinAddr]

.unreq pinAddr

```

```

.unreq pinVal
mov      pc, lr
//-----
//- GPIOPinRead
//-
//- resets the GPIO Pin to off
//- Parameters:
//-      r0: GPIO Pin number
//-      r1: Value to set
//-
//- Returns:
//-      r0: {
//-          (pin is non-zero, #1),
//-          (pin is zero, #0)
//-      }
//-----
.globl GPIOPinRead
GPIOPinRead:
pinAddr .req r0
pinVal  .req r1

ldr     r2, [pinAddr]
and     r2, r1
cmp     r2, #0
moveq   r0, #0
movne   r0, #1

.unreq pinAddr
.unreq pinVal

mov     pc, lr

```

9.1.5 main.s

```

//-----
//- main.s rEOS Main Function
//-
//- Kicks off the rEOS
//
// Copyright (c) 2016 by James J. Perry
//-----
.section .init
.globl _start

```

```

_start:
b          main
.section .text

main:

mov        sp, #0x8000          // to top of stack.

bl         initREOS
.globl _end

_end:

stop:      b stop                // Loop forever when idle.

```

9.1.6 mpce.s

```

//-----
// mpce.s
//
// Project:  rEOS 0.1
//
// Description:  This module contains the base
// code for the process scheduler.
//
// Copyright (c) 2015, 2014 by James J. Perry
//-----
// - Equates for Process Control Block. These
// - represent the offset of the field from the
// - beginning of the control block
//-----

.equ PCB_base,      0x00000000    // The start of the PCB
.equ PCB_pid,       0x00000000    // unique process id
.equ PCB_state,     0x00000004    // current process state )
.equ PCB_pspace,    0x00000008    // address of pspace
.equ PCB_psl,       0x0000000C    // length of pspace in bytes
.equ PCB_sspace,    0x00000010    // address of sspace
.equ PCB_spl,       0x00000014    // Length of sspace in bytes
.equ PCB_lpc,       0x00000018    // lpc

```

```

.equ PCB_lsp,          0x0000001c    // lsp
.equ PCB_cycles,      0x00000020    // #of VM cycles before swap.
.equ PCB_source,      0x00000024    // Status read buffer
.equ PCB_target,      0x00000028    // Status write buffer
.equ PCB_next,        0x0000002c    // Next PCB
.equ PCB_prev,        0x00000030    // Previous PCB
//-----
//- State Equates
//-----
.equ PCB_STATE_new,          0x00000000
.equ PCB_STATE_ready,       0x00000001
.equ PCB_STATE_running,     0x00000002
.equ PCB_STATE_waiting,     0x00000003
.equ PCB_STATE_terminated,  0x00000004
//-----
//- Scheduler Mode Equates
//-----
.equ SCH_STATE_SPE,         0x00000000
.equ SCH_STATE_MPCE,       0x00000001
.equ SCH_STATE_queue,      0x00000000
.equ SCH_STATE_stack,      0x00000001
//-----
//- PCB Return Codes
//-----
.equ PCB_RC_Success,       0x00000000
//-----
//- Scheduler Populate Queue (MPCE)
//-
//- This function is called by the start-up code to
//- define the initial processes to place in the
//- ready stack. A separate call is required for
//- each process. The process is placed in the
//- FIFO ready queue. Note the PBC of all incoming
//- processes must have pbc_next set to 0.
//-
//- Parameters:
//-     r0 : PCB of Process.
//-
//-----
.globl schedulerMPCEAdd
schedulerMPCEAdd:

new      .req      r0
lastPtr  .req      r1

```

```

currVal    .req          r2
currPtr    .req          r3
    // if compare last and null
ldr        currPtr, =schedReadyCurr
ldr        currVal, [currPtr]
cmp        currVal, #0

    // Case 1: Empty Queue
streq     new, [currPtr]
ldreq     lastPtr, =schedReadyLast
streq     new, [lastPtr]
moveq     pc, lr

lastVal    .req          r2
    // Case 2:
ldr        lastPtr, =schedReadyLast
ldr        lastVal, [lastPtr]
add        lastVal, #PCB_next
str        new, [lastVal]
str        new, [lastPtr]
ldr        lastVal, [lastPtr]
add        lastVal, #PCB_next
str        currVal, [lastVal]
mov        pc, lr

.unreq    currVal
.unreq    currPtr
.unreq    new
.unreq    lastPtr
.unreq    lastVal
//-----
// - MPCE Scheduler Run
// -
// - This function is called by the start-up code
// - after all initial processes are added to the
// - Scheduler ready queue. It manages all process
// - scheduling.
// -
// - Parameters:
// -   r0 :   Quantum setting
//-----
.global schedulerMPCERun

```

schedulerMPCERun:

```

quantum    .req r5
curr       .req r0
field      .req r2
last       .req r1
next       .req r3
push       {r5, lr}
           // Get PCB of next process to run.
mov        quantum, r0

```

schedulerMPCERun_loop:

```

ldr        curr, =schedReadyCurr    // curr<-&schedReadyCurr
ldr        curr, [curr]              // curr<-@Current PCB
           // while curr != null
cmp        curr, #0
           // if it is null exit
popcq     {lr}
moveq     pc, lr

ldr        curr, =schedReadyCurr    // curr<-&schedReadyCurr
ldr        curr, [curr]              // curr<-@Current PCB
           // Point field to state
mov        field, curr               // field <- @current PCB
add        field, #PCB_state         // field
           // <- @current PCB.PCB_state

ldr        next, [field]
mov        r1, #PCB_STATE_terminated
cmp        next, r1
beq        nextProcess

           // Set current state to running
mov        r1, #PCB_STATE_running    // r1 = running state
str        r1, [field]               // PCB.PCB_state<- running
           // Set cycles to quantum
mov        field, curr               // field <- @current PCB
add        field, #PCB_cycles         // field
           // <- @current PCB.PCB_cycles
str        quantum, [field]

bl        VM_main
           // Update process to terminated and set next process
nextProcess:
ldr        curr, =schedReadyCurr    // curr<-&schedReadyCurr
ldr        field, [curr]             // field<-&Current PCB
add        field, #PCB_next          // field<-

```

```

                                //      &Current PCB.PCB_next
ldr      next, [field]          // next<-&next process PCB
str      next, [curr]
mov      curr, next
b        schedulerMPCERun_loop

.unreq   quantum
.unreq   curr
.unreq   ield
.unreq   last
.unreq   next

```

9.1.7 processmgr.s

```

//-----
// processmgrbase.s
//
// Project:  rEOS 0.1
//
// Description:  This module contains the base
// code for the process scheduler.
//
// Copyright (c) 2015, 2014 by James J. Perry
//-----
// - Equates for Process Control Block. These
// - represent the offset of the field from the
// - beginning of the control block
//-----
.equ PCB_base,      0x00000000    // PCB Start
.equ PCB_pid,      0x00000000    // unique process id
.equ PCB_state,    0x00000004    // process state
.equ PCB_pspace,   0x00000008    // pspace address
.equ PCB_psl,      0x0000000C    // pspace length
.equ PCB_sspace,   0x00000010    // sspace address
.equ PCB_spl,      0x00000014    // sspace length
.equ PCB_lpc,      0x00000018    // lpc
.equ PCB_lsp,      0x0000001C    // lsp
.equ PCB_cycles,   0x00000020    // VM cycles before swap
.equ PCB_source,   0x00000024    // read buffer PCB
.equ PCB_target,   0x00000028    // write buffer PCB
.equ PCB_next,     0x0000002c    // Next PCB
.equ PCB_prev,     0x00000030    // Previous PCB
//-----

```

```

//- State Equates
//-----
.equ PCB_STATE_new,          0x00000000
.equ PCB_STATE_ready,       0x00000001
.equ PCB_STATE_running,     0x00000002
.equ PCB_STATE_waiting,     0x00000003
.equ PCB_STATE_terminated,  0x00000004
//-----
//- Scheduler Mode Equates
//-----
.equ SCH_STATE_SPE,         0x00000000
.equ SCH_STATE_MPCE,       0x00000001
.equ SCH_STATE_queue,      0x00000000
.equ SCH_STATE_stack,      0x00000001
//-----
//- PCB Return Codes
//-----
.equ PCB_RC_Success,       0x00000000

.section .data
.align 2
schedMode:          .int          0xFFFFFFFF
.globl schedReadyCurr
schedReadyCurr:    .int          0x00000000
.globl schedReadyLast
schedReadyLast:    .int          0x00000000
.globl schedDS
schedDS:           .int          0x00000000
.globl schedQuantum
schedQuantum:     .int          0x00000000

.section .text
//-----
// Set Quantum
//
// Parameters:
//     r0 : quantum
//-----
.globl setQuantum
setQuantum:
q          .req          r0
ldr        r1, =schedMode
str        r0, [r1]
mov        pc, lr

```

```

.unreq    q
//-----
//    Set Scheduler Mode
//
// - This function is called by the start-up code to
// - define the scheduling mode.
// - Parameters:
// -    r0 : mode
// - Return:
// -    fails silently
//-----

.global schedulerSetMode
schedulerSetMode:
sMode    .req    r1
ldr      sMode, =schedMode
cmp      r0, #SCH_STATE_SPE
streq   r0, [sMode]
cmp      r0, #SCH_STATE_MPCE
streq   r0, [sMode]
mov     pc, lr
.unreq   sMode
//-----
//    Set Scheduler Mode
//
// - This function is called by the start-up code to
// - define the scheduling mode.
// - Parameters:
// -    r0 : mode
// - Return:
// -    fails silently
//-----

.global schedulerSetDS
schedulerSetDS:
sMode    .req    r1
ldr      sMode, =schedDS
cmp      r0, #SCH_STATE_stack
streq   r0, [sMode]
cmp      r0, #SCH_STATE_queue
streq   r0, [sMode]
mov     pc, lr
.unreq   sMode
//-----
// - Scheduler SPE
//

```

```

//- This function is called by the start-up code to
//- define the initial processes to place in the
//- ready stack. A separate call is required for each
//- process. The process is placed in the LIFO ready
//- queue. Note the PBC of all incoming processes must
//- have pbc_next set to 0.
//-
//-
//- Parameters:
//-     r0 : PCB of Process.
//-
//-----
.globl schedulerSPE
schedulerSPE:
new         .req r0
mode        .req r1

push        {lr}
           // if compare currVal and null
ldr         mode, =schedDS
ldr         mode, [mode]           // temp<-&curr pcb
push        {mode}
cmp         mode, #0               // if empty queue
bleq        schedulerSPEAddQueue
pop         {mode}
cmp         mode, #1
bleq        schedulerSPEAddStack
pop         {lr}
mov         pc, lr
.unreq     new
.unreq     mode
//-----
//- Scheduler Populate Stack SPCE
//-
//- This function is called by the start-up code to
//- define the initial processes to place in the ready
//- stack. A separate call is required for each process.
//- The process is placed in the LIFO ready queue.
//- Note the PBC of all incoming processes must have
//- pbc_next set to 0.
//-
//- Parameters:
//-     r0 : PCB of Process.
//-

```

```

//-----
.globl schedulerSPEAddStack
schedulerSPEAddStack:
new      .req      r0
currPtr  .req      r1
currVal  .req      r2
next     .req      r3

        // if compare currVal and null
ldr      currPtr, =schedReadyCurr// curr<-&schedReadyLast
ldr      currVal, [currPtr]          // temp<-&curr pcb
cmp      currVal, #0                // if empty queue
        // new.pcb_next<-currval
movne    next, new
addne    next, #PCB_next
strne    currVal, [next]            // new.PCB_next<
        // curr = r0
str      new, [currPtr]
mov      pc, lr
.unreq   new
.unreq   currPtr
.unreq   currVal
.unreq   next

//-----
//- Scheduler Populate Queue
//-
//- This function is called by the start-up code to
//- define the initial processes to place in the ready
//- stack. A seperate call is required for each process.
//- The process is placed in the FIFO ready queue.
//- Note the PBC of all incoming processes must have
//- pbc_next set to 0.
//-
//- Parameters:
//-      r0 : PCB of Process.
//-
//-----
.globl schedulerSPEAddQueue
schedulerSPEAddQueue:
new      .req      r0
lastPtr  .req      r1
currVal  .req      r2
currPtr  .req      r3

```

```

ldr      currPtr, =schedReadyCurr
ldr      currVal, [currPtr]
cmp      currVal, #0

        // Case 1:
streq    new, [currPtr]
ldreq    lastPtr, =schedReadyLast
streq    new, [lastPtr]
moveq    pc, lr

lastVal  .req r2
        // Case 2:
ldr      lastPtr, =schedReadyLast
ldr      lastVal, [lastPtr]
add      lastVal, #PCB_next
str      new, [lastVal]
str      new, [lastPtr]
mov      pc, lr
.unreq   currVal
.unreq   currPtr
.unreq   new
.unreq   lastPtr
.unreq   lastVal

//-----
//- SPE Scheduler Run
//-
//- This function is called by the start-up code
//- after all resident processes are added to the
//- Scheduler ready queue. It manages all
//- process scheduling.
//-
//- Parameters:
//-     None.
//-
//-----
--

.global schedulerSPERun
schedulerSPERun:

curr      .req r0
field     .req r2
next      .req r3

```

```

push    {lr}
        // Get PCB of next process to run.

ldr     curr, =schedReadyCurr    // curr<-&schedReadyCurr
ldr     curr, [curr]             // curr<-@Current PCB
schedulerSPERun_loop:
        // while curr != null
cmp     curr, #0
        // if it is null exit
popseq  {lr}
moveq   pc, lr
        // Point field to state
mov     field, curr
add     field, #PCB_state

        // Set current state to running
mov     r1, #PCB_STATE_running
str     r1, [field]
        // Set cycles to -1 (run till end)
mov     field, curr
add     field, #PCB_cycles
mov     r1, #0xFFFFFFFF
str     r1, [field]

bl      VM_main
        // Update process to terminated and set next process
ldr     curr, =schedReadyCurr
ldr     field, [curr]
add     field, #PCB_state
mov     r1, #PCB_STATE_terminated
str     r1, [field]

ldr     field, [curr]
add     field, #PCB_next
ldr     next, [field]

str     next, [curr]
mov     curr, next

bl      schedulerSPERun_loop

```

9.1.8 processmsg.s

```

//-----
// processmsg.s
//
// Project: rEOS 0.1
//
// Description: This module contains the base
// code for the process scheduler.
//
// Copyright (c) 2016, 2015 by James J. Perry
//-----
// Equates for Process Control Block. These
// represent the offset of the field from the
// beginning of the control block
//-----

.equ    PCB_base,      0x00000000    // The start of the PCB
.equ    PCB_pid,      0x00000000    // unique process id
.equ    PCB_state,    0x00000004    // current process state
.equ    PCB_pspace,   0x00000008    // address of pspace
.equ    PCB_psl,      0x0000000C    // length of pspace
.equ    PCB_sspace,   0x00000010    // address of sspace
.equ    PCB_spl,      0x00000014    // Length of the sspace
.equ    PCB_lpc,      0x00000018    // lpc
.equ    PCB_lsp,      0x0000001C    // lsp
.equ    PCB_cycles,   0x00000020    // VM cycles before swap
.equ    PCB_source,   0x00000024    // read buffer PCB
.equ    PCB_target,   0x00000028    // write buffer PCB
.equ    PCB_next,     0x0000002c    // Next PCB in
.equ    PCB_prev,     0x00000030    // Previous PCB
//-----
// State Equates
//-----
.equ    PCB_STATE_new,      0x00000000
.equ    PCB_STATE_ready,    0x00000001
.equ    PCB_STATE_running,  0x00000002
.equ    PCB_STATE_waiting,  0x00000003
.equ    PCB_STATE_terminated, 0x00000004

//-----
// Lock Equates
//-----

```

```

.equ      MSG_free,          0x00000000
.equ      MSG_unwritten,    0x00000001
.equ      MSG_written,      0x00000002
//-----
// - PCB Return Codes
//-----
.equ      PMSG_RC_Success,   0x00000000
.equ      PMSG_RC_samePID,   0xffffffff
.equ      PMSG_RC_notUNWritten, 0xffffffffc
.equ      PMSG_RC_notWritten, 0xffffffffb
.equ      PMSG_RC_PIDnotFound, 0xFFFFFFFF

.section .data
.align 2

.section .text
//-----
// msgOpenChk
//
// Function to determine if a pid exists in the
// waiting queue.
//
// Parameters:
//     r0 : pidNumber
// Returns:
//     r0 : status    {
//                     (pid not found, 0xffffffff),
//                     (pid.pcb_target!=0, 0xffffffffe)
//                     (else, 0x00000000)
//                     }
//     r1 :           {
//                     (r0==0, &pid.pcb_tgt)
//                     (r0!=0, garbage)
//                     }
//-----
.globl msgOpenChk
msgOpenChk:
tgtPID    .req r0
runPCB    .req r1
currPCB   .req r2
currPID   .req r3
          // point to running pcb;
ldr  currPCB, =schedReadyCurr
mov  runPCB, currPCB

```

```

ldr      currPID, [currPCB]
cmp      tgtPID, currPID

bne      pmsg_MO_ne
mov      r0, #PMSG_RC_samePID
mov      pc, lr
pmsg_MO_ne:
add      currPCB, #PCB_next
ldr      currPCB, [currPCB]
cmp      currPCB, runPCB
bne      psmg_MO_cont
mov      r0, #PMSG_RC_PIDnotFound
mov      pc, lr
psmg_MO_cont:
ldr      currPID, [currPCB]
cmp      tgtPID, currPID
bne      pmsg_MO_ne

                // The pcb specified.
.unreq   tgtPID
src      .req r0

add      currPCB, #PCB_source
ldr      src, [currPCB]
cmp      src, #0
beq      pmsg_MO_good
mov      r0, #0xFFFFFFFF
mov      pc, lr
pmsg_MO_good:
                // curr.pcb_source_target<-recPID
mov      r0, #0
mov      r1, currPCB
mov      pc, lr

.unreq   src
.unreq   currPID
.unreq   currPCB
.unreq   runPCB
//-----
//  msgOpenChkLock
//
//  Function to Check if queue is locked
//
//  Parameters:

```

```

//      r0 : Source Data Pointer
// Returns:
//      r1 : {
//              (locked, 1),
//              (open, 0)
//      }
//-----
.globl msgOpenChkLock
msgOpenChkLock:

data .req          r2
mov     r1, #0
ldr     data, [r0]
cmp     data, #MSG_free
movne   r1, #1
mov     pc,lr

//-----
// msgOpenLock
//
// Function to lock the queue from reading
//
// Parameters:
//      r0 : Source Data Pointer
//      r1 : Buffer size
// Returns:
//      none
//-----
.globl msgOpenLock
msgOpenLock:

data .req r2

mov     data, #MSG_unwritten
str     data, [r0]
add     r0, #4
str     r1, [r0]
mov     pc,lr

//-----
// msgOpen
//
// Function to determine if a pid exists in the
// waiting queue.

```

```

//
// Parameters:
//     r0 : SourcePCB
//     r1 : TargetPCB
// Returns:
//     none
//-----
.globl msgOpenDone
msgOpenDone:
src      .req      r0
tgt      .req      r1
srcPID   .req      r2
tgtPID   .req      r3

ldr      srcPID, [src]
ldr      tgtPID, [tgt]

add      src, #PCB_target
str      tgtPID, [src]

add      tgt, #PCB_source
str      srcPID, [tgt]
mov      pc, lr
//-----
// msgWrite
//
// Function to Check if queue is locked
//
// Parameters:
//     r0 : Message Buffer Address
//     r1 : Number of words to move
//     r2 : input buffer
// Returns:
//
//-----
.globl msgWrite
msgWrite:
buffAddr .req      r4
msgSize  .req      r5
tgtAddr  .req      r6
temp     .req      r2
        // is buffer in unwritten state?
push     {r4, r5, r6}

```

```

mov     buffAddr, r0
mov     msgSize, r1
mov     tgtAddr, r2
ldr     temp, [buffAddr]
cmp     temp, #MSG_unwritten
beq     msgWriteCheckReceiver
mov     r0, #PMSG_RC_notUNWritten
pop     {r4, r5, r6}
mov     pc, lr

```

// is receiver valid?

msgWriteCheckReceiver:

```

ldr     temp, =schedReadyCurr
ldr     temp, [temp]
add     temp, #PCB_target
ldr     r0, [temp]
bl     msgOpenChk
cmp     r0, #PMSG_RC_PIDnotFound
bne     msgWriteSetSize
mov     r0, #PMSG_RC_PIDnotFound
pop     {r4, r5, r6}
mov     pc, lr

```

// Set Size and write status

msgWriteSetSize:

```

mov     r0, #MSG_free
str     r0, [buffAddr]
add     buffAddr, #4
str     msgSize, [buffAddr]
add     buffAddr, #4

```

// Start Copy

msgWriteCopy:

```

cmp     msgSize, #0
beq     msgWriteExit
mov     temp, buffAddr
ldr     temp, [temp]
str     temp, [tgtaddr]
add     buffAddr, #4
add     tgtAddr, #4
sub     msgSize, #4
b     msgWriteCopy

```

msgWriteExit:

```

pop      {r4, r5, r6}
mov      pc, lr

//-----
//  msgRead
//
//
//  Parameters:
//      r0 : Message Buffer Address
//      r1 : input buffer
//  Returns:
//
//-----

.globl msgRead
msgRead:

srcAddr  .req      r4
msgSize  .req      r5
tgtAddr  .req      r6
temp     .req      r2
n        .req      r0
        // is buffer in written state?
push     {r4, r5, r6}
mov      tgtAddr, r0

        // is sender valid?
ldr      temp, =schedReadyCurr
ldr      temp, [temp]
add      temp, #PCB_source
ldr      r0, [temp]
bl       msgOpenChk
cmp      r0, #PMSG_RC_PIDnotFound
bne      msgReadCheckSender
mov      r0, #msgReadSetSize
pop      {r4, r5, r6}
mov      pc, lr

msgReadCheckSender:
ldr      temp, [srcAddr]
cmp      temp, #MSG_written
beq      msgReadCheckSender
mov      r0, #PMSG_RC_notWritten
pop      {r4, r5, r6}

```

```

mov     pc, lr
// Set Size and read status
msgReadSetSize:
mov     r0, #MSG_written
str     r0, [srcAddr]
add     srcAddr, #4
str     msgSize, [srcAddr]
mov     n, msgSize
add     srcAddr, #4

// Start Copy
msgReadCopy:
cmp     msgSize, #0
beq     msgReadExit
mov     temp, srcAddr
ldr     temp, [temp]
str     temp, [tgtaddr]
add     srcAddr, #4
add     tgtAddr, #4
sub     msgSize, #4
b       msgReadCopy

msgReadExit:
mov     r0, #0
pop     {r4, r5, r6}
mov     pc, lr

```

9.1.9 string.s

```

//-----
// string.s
//
// Project:  rEOS 0.1
//
// Description:  This module contains the code
// handle and display strings.
//
// Copyright (c) 2015, 2014 by James J. Perry
//-----

.section .text
//-----
// - drawString
// -

```

```

//- First word of string is length, subsequent words
//- are characters
//- Parameters:
//-     r0 : address of string
//-     r1 : contains the ULHC x coord to draw to
//-     r2 : contains the ULHC y coord to draw to
//-
//-     Fails silently on illegal write or
//-     unsupported ASCII code.
//-----
.globl drawString
drawString:

push        {r4,r5,r6,r7,lr}

nextChar    .req r4
nextX       .req r5
nextY       .req r6
charLeft    .req r7

ldr         charLeft, [r0]
mov         nextX, r1
mov         nextY, r2
mov         nextChar, r0

STRING_DRAWSTRING_next:
cmp         charLeft, #0
beq         STRING_DRAWSTRING_exit
blt         STRING_DRAWSTRING_exit
add         nextChar,#4
ldr         r0,[nextChar]
mov         r1, nextX
mov         r2, nextY
bl         drawCharacter
sub         charLeft,#1
add         nextX, #8
b          STRING_DRAWSTRING_next

STRING_DRAWSTRING_exit:
pop        {r4,r5,r6,r7, lr}
mov        pc, lr

```

9.1.10 time.s

```

//-----
//- vm.s support for rEOS VMR
//
// Copyright (c) 2015, 2014 by James J. Perry
//-----
//- Equates
//-----
.equ      TIMER_addr_$time,    0x20003000
//-----
//- getTimerBase
//-
//- Function to get the start address of the system
//- timer block
//- Parameters:
//-     None.
//- Return values:
//-     r0 : contians the start address of system timer
//-
//-----
.globl getTimerBase
getTimerBase:
ldr      r0,=TIMER_addr_$time
mov      pc, lr

//-----
//- getTimeStamp
//-
//- Function to get the current value of the Timer
//- Parameters:
//-     None.
//- Return values:
//-     r0,r1 : returns value of eight byte system
//-     time counter.
//-
//-----
.globl getTimeStamp
getTimeStamp:
ldr      r0,=TIMER_addr_$time
ldrd    r0,r1,[ r0, #4]
mov      pc, lr

```

```

//-----
//- waitForTimer
//-
//- Function wait for a predetermined amount of time.
//-
//- Note: This function only guarantees that at least
//-       n milliseconds will pass during the wait.
//- Parameters:
//-       r0:   number of milliseconds to wait.
//- Return values:
//-       none
//-----
.globl waitForTimer
waitForTimer:
msDelay    .req r2

mov        msDelay, r0           // Save the ms to delay.
push      {lr}
bl        getTimeStamp         // get time now into r0, r1

start      .req r3              // copy lower 4 bytes from r3

mov        start, r0
loop1_getTimeSamp$:
bl        getTimeStamp         // loop until time passed.
elapsed   .req r1
sub        elapsed, r0, start
cmp        elapsed, msDelay
bls       loop1_getTimeSamp$
pop       {pc}

.unreq    elapsed
.unreq    msDelay
.unreq    start

```

9.1.11 utils

```

//-----
//  util.s
//
//  Project:  rEOS 0.1

```

```

//
// Description: This module contains general utility
//              functions.
//
// Copyright (c) 2015, 2014 by James J. Perry
//-----
//-----
// - Equates Common values
//-----
.global zero
.global one
.global two
.global three
.global four
.global five

.equ    zero,    0x00000000
.equ    one,     0x00000001
.equ    two,     0x00000010
.equ    three,  0x00000011
.equ    four,   0x00000100
.equ    five,   0x00000101
//-----
// - clear Memory
// -
// - Function to clear blocks of memory to zero.
// - Parameters:
// -     r0 Start address to clear
// -     r1 number of 32-bit words to clear
// - Return values:
// -     none.
// -
//-----
.global clearMem
clearMem:

startAddr .req r0
count     .req r1
z         .req r2
clearMemLoop:
cmp      count, #zero
beq      clearMemDone
streq   z, [startAddr]
add     startAddr, #4

```

```

sub     count,#4
b       clearMemLoop
clearMemDone:
mov     pc, lr

//-----
// - division
// -
// - solves a=nq +r division for q
// - Parameters:
// -     r0 a
// -     r1 n
// - Return values:
// -     r0 q
// -
// - Note: This code does not account for n <=0
//-----
.globl div
div:
a       .req r0
n       .req r1
q       .req r2
mov     q,#0
util_div_loop:
cmp     a,n
blt     util_div_exit
sub     a,n
add     q, #1
b       util_div_loop
util_div_exit:
mov     r0, q
mov     pc, lr
.unreq  a
.unreq  n
.unreq  q

//-----
// - mod
// -
// - solves a=nq +r division for r
// - Parameters:
// -     r0 a
// -     r1 n
// - Return values:

```

```

//-      r0 r
//-
//- Note: This code does not account for n <=0
//-----
.globl mod
mod:
a          .req r0
n          .req r1
r          .req r0
util_mod_loop:
cmp        r,n                // if (r<n)
blt        util_mod_exit
sub        r,n
b          util_mod_loop

util_mod_exit:
mov        pc, lr

.unreq    a
.unreq    n
.unreq    r

```

9.1.12 vid.s

```

//-----
//- vid.s Core Video support for rEOS VMR
//
// Copyright (c) 2015, 2014 by James J. Perry
//-----
//- Equates
//-----
.equ      VID_mailbox_base,      0x2000B880
.equ      VID_channel_read,      0x2000B880
.equ      VID_channel_status,    0x2000B898
.equ      VID_channel_write,     0x2000B8A0

        // Mailbox statuses
.equ      VID_write_ok,          0x80000000
.equ      VID_read_ok,           0x40000000

        // GPUinstructions
.equ      VID_GPU_flushCache,    0x40000000

```

```

        // Frame buffer offsets
.equ VID_framebuffer_pWidth,      0x00000000
.equ VID_framebuffer_pHeight,     0x00000004
.equ VID_framebuffer_vWidth,      0x00000008
.equ VID_framebuffer_vHeight,     0x0000000C
.equ VID_framebuffer_GPU_pitch,   0x00000010
.equ VID_framebuffer_bitDepth,    0x00000014
.equ VID_framebuffer_x,           0x00000018
.equ VID_framebuffer_y,           0x0000001C
.equ VID_framebuffer_GPU_ptr,     0x00000020
.equ VID_framebuffer_GPU_size,    0x00000024

        // Frame Limits
                // Width and height maxfor video frame.
.equ VID_frame_side_max,          0x00001000
                // Bit Depth maxfor a frame.
.equ VID_frame_bit_max,           0x00000020

        // Return Codes
.equ VID_frame_mailboxWrite_fail,0x00000001
//-----
// - vidWriteMailBox
// -
// - Support write to video
// - Parameters:
// -     r0 data to write
// -     r1 mailbox number
// -
// - Return values:
// -     r0 returns 1 on failure
// -
// - Notes:
// -     r0 should set bit 30 to indicate cache flush
//-----
.globl vidMailBoxWrite
vidMailBoxWrite:

msg         .req r0                // The data to write
channel    .req r1                // mailbox channel to
                                        // write to

//-----
// - Checking for valid request

```

```

        //-          Fails return immediately, but silently
        //-----
tst      msg, #0b1111          // are low 4 bits off
movne    pc, lr              // invalid low nibble
cmp      channel, #15        // check mailbox
movhi    pc, lr              // invalid low nibble

        //-----
        //- Compute Write Value
        //-----
add      msg, channel

        //-----
        //- Poll until write allowed
        //-----
writeMailBox .req      r2
status      .req      r3

ldr      writeMailBox, =VID_channel_write
vidWriteWait:
ldr      status, =VID_channel_status
tst      status, #VID_write_ok
bne      vidWriteWait

        //-----
        //- Write data
        //-----
str      msg, [writeMailBox]
mov      pc, lr
.unreq   status
.unreq   writeMailBox
.unreq   channel
.unreq   msg

        //-----
        //- vidReadMailBox
        //-
        //- Support Read Mailbox
        //- Parameters:
        //-      r0: mailbox number
        //- Return values:
        //-      r0: message
        //-
        //-----

```

```

.globl vidMailBoxRead
vidMailBoxRead:
    //-----
    // Validate input: is specified mailbox real?
    //-----
reqBox    .req r0           // channel of the box
cmp       reqBox, #15      // box number must be 0-F
movhi    pc,lr            // exit on bad box.

    //-----
    // Setup read channel
    //-----
readMailBox .req r1       // r1 address of read box
ldr       readMailBox, =VID_channel_read

    //-----
    // Loop until the read status bit is set
    //-----
status    .req r2

vidReadWait:
ldr       status, =VID_channel_status // get status
tst      status, #VID_read_ok        // test bit 30
bne      vidReadWait

    //-----
    // Determine if our message loop back if not.
    //-----
msg       .req r2           // message
ldr       msg, [readMailBox] // load message
inChannel .req r3           // r3 msg channel
and       inChannel, msg, #0b1111 // determine channel
teq      inChannel, reqBox

bne      vidReadWait

    //-----
    // return msg
    //-----
rmsg     .req r0
and      rmsg, msg, #0xffffffff // Strip channel info
mov      pc, lr
.unreq   rmsg
.unreq   msg
.unreq   status
.unreq   reqBox

```

```

//-----
//- vidSetFramebuffer
//-
//- Frame buffer setup
//- Parameters:
//-     r0: width
//-     r1: height
//-     r2: pixel depth
//- Return values:
//-     r0: 0,          failed operation
//-          pointer,    frame buffer info.
//-----
.section .text
.globl vidSetFramebuffer
vidSetFramebuffer:
width     .req r0
height    .req r1
depth     .req r2

        //-----
        // size or depth is too large, then return 0
        //-----
cmp      width, #VID_frame_side_max
cmpls   height, #VID_frame_side_max
cmpls   depth, #VID_frame_bit_max
movhi   r0, #0
movhi   pc, lr

        //-----
        // Construct the buffer request
        //-----

fbInfoAddr     .req r4

push    {r4, lr}
ldr     fbInfoAddr, =frameBufferInfo
str     width, [fbInfoAddr, #VID_framebuffer_pWidth]
str     height, [fbInfoAddr, #VID_framebuffer_pHeight]
str     width, [fbInfoAddr, #VID_framebuffer_vWidth]
str     height, [fbInfoAddr, #VID_framebuffer_vHeight]
str     depth, [fbInfoAddr, #VID_framebuffer_bitDepth]
.unreq  width
.unreq  height

```

```

.unreq    depth

mov       r0, fbInfoAddr
add       r0, #VID_GPU_flushCache
mov       r1, #1

bl        vidMailBoxWrite

//-----
// Prepare and read
//-----
mov       r0, #1
bl        vidMailBoxRead

//-----
// return 0 on failure
//-----
result    .req r0
teq       result, #0

movne     result, #0
popne     {r4,lr}
//-----
// else return infor buffer
//-----
mov       r0, fbInfoAddr
pop       {r4, pc}

.unreq    fbInfoAddr
.unreq    result

//-----
// framebuffer
//-----

.section .data
.align 12           // Force a word boundary

.globl framebufferInfo
frameBufferInfo:
.space 40           // Reserve framebuffer space

```

9.1.13 vm.s

```

//-----
//- vm.s support for rEOS VMR
//
// Copyright (c) 2015, 2014 by James J. Perry
//-----
//- Equates
//-----
//-----Instructions
//
//      Note:      the constants assigned to the rEOS
//                  VMR instructions
//                  have no meaning beyond being unique.
//
//-----
.equ VM_add,          0x0000000B    // add top values
.equ VM_and,          0x0000000E    // andtop values
.equ VM_asl,          0x00000017    // shift left logically
.equ VM_asr,          0x00000018    // shift right arithmetic
.equ VM_bin2str,      0x00000013    // convert binary to string
.equ VM_call,         0x0000001D    // linkage call
.equ VM_dec,          0x00000009    // decrement the top value
.equ VM_div,          0x00000019    // division
.equ VM_drawCharacter, 0x00000003    // draw a character
.equ VM_drawLine,     0x0000001A    // draw a line
.equ VM_drawPixel,    0x00000004    // drawPixel
.equ VM_drawString,   0x00000014    // drawString
.equ VM_dump,         0x00000002    // abort and dump
.equ VM_dup,          0x00000006    // duplicate.
.equ VM_fill,         0x0000001B    // fill rect
.equ VM_gpioRead,     0x0000002A    // gpio_read
.equ VM_gpioReadSet,  0x0000002B    // gpio_read_setup
.equ VM_gpioWrite,    0x0000002C    // gpio_read_set
.equ VM_halt,         0x00000005    // end execution
.equ VM_inc,          0x0000000A    // increment
.equ VM_jump,         0x0000001F    // unconditional jump
.equ VM_jmpeq,        0x00000020    // jump to if ==
.equ VM_jmpgt,        0x00000022    // jump to id >
.equ VM_jmpgte,       0x00000023    // jump to id >=
.equ VM_jmpneq,       0x00000021    // jump to if !=
.equ VM_lsr,          0x00000024    // logical shift right
.equ VM_mod,          0x00000025    // modulus

```

```

.equ VM_msgOpen,          0x00000027    // open msg buffer.
.equ VM_msgRead,         0x00000029    // read msg buffer.
.equ VM_msgWrite,       0x00000028    // write to msg buffer.
.equ VM_mul,            0x00000015    // multiply
.equ VM_neg,            0x00000012    // negate
.equ VM_nop,            0x00000007    // no operation
.equ VM_not,            0x00000011    // not
.equ VM_or,             0x0000000F    // or
.equ VM_pause,         0x00000026    // pause
.equ VM_pop,            0x00000008    // pop
.equ VM_psh,            0x00000001    // push
.equ VM_ret,            0x0000001E    // linkage ret
.equ VM_ror,            0x0000001C    // ror
.equ VM_sub,            0x0000000C    // subtract
.equ VM_swap,           0x00000010    // swap
.equ VM_time,           0x0000000D    // get time stamp
.equ VM_xor,            0x00000016    // xor
.equ VM_maxInstruct,    0x0000002D    // wordcode < this
//-----
//-----Utility values
//-----
.equ VM_wordSize,       0x00000004    // word size
.equ VM_dWordSize,     0x00000008    // double word size
//-----
//-----Dump values
//-----
.equ VM_DUMP_InstNotFnd, 0x00000031    // WC not found;
.equ VM_DUMP_PSBad,     0x00000032    // PC past pSpace
.equ VM_DUMP_USR,       0x00000033    // User initiated
.equ VM_DUMP_NOP,       0x00000034    // User initiated
.equ VM_DUMP_SPBad_aloc, 0x00000035    // SP fail on aloc
.equ VM_DUMP_SPBad_dloc, 0x00000036    // SP fail on dloc
.equ VM_DUMP_InstBad,   0x00000037    // WC not valid range.
.equ VM_DUMP_SPBad_dloc_x, 0x00000038 // SP fail on dloc_x

//-----
//-- State Equates
//-----

.equ PCB_STATE_new,     0x00000000
.equ PCB_STATE_ready,  0x00000001
.equ PCB_STATE_running, 0x00000002
.equ PCB_STATE_waiting, 0x00000003
.equ PCB_STATE_terminated, 0x00000004

```

```

//-----
// - VM Data Area
//-----
.section .data
.align 2

.align 1
.globl VM_Dump_On_NOP
VM_Dump_On_NOP: .int 0x00000001
.globl VM_fail_on_user
VM_fail_on_user: .int 0x00000001

lpc      .req r4
lsp      .req r12
psl      .req r5
spl      .req r6
pspace   .req r7
sspace   .req r8
vmopc    .req r9
vmosp    .req r10
vmopc    .req r11
ccntr    .req r3
//-----
// VM_dloc
//
// This function changes the top of the stack to
// be one position down (that // is vmosp += word-size).
// The rEOS VMR uses this function to ensure correct
// increments of the lsp, to calculate vmosp and to
// protect memory
//-----
VM_dloc:
add      lsp, #VM_wordSize      // dloc 1 word
cmp      lsp, spl
movhi    r0, #VM_DUMP_SPBad_dloc
bhi     VM_dumpAbort
mov      vmosp, lsp
add      vmosp, sspace
mov      pc, lr

//-----
// VM_dloc_x
//

```

```

// This function changes the top of the stack to
// be x positions down (that // is vmisp += word-size).
// The rEOS VMR uses this function to ensure correct
// increments of the lsp, to calculate vmisp and to
// protect memory
//
// Registers:
//      r0:  number of words to pop off the stack.
//-----
VM_dloc_x:
mov     r1, #VM_wordSize
mul     r0, r1
add     lsp, r0                // dloc x words
cmp     lsp, spl
movhi   r0, #VM_DUMP_SPBad_dloc_x
bhi     VM_dumpAbort
mov     vmisp, lsp
add     vmisp, sspace
mov     pc, lr

//-----
// VM_aloc
//
// This function changes the top of the stack to
// be one position up (that is vmisp -= word-size).
// The rEOS VMR uses this function to ensure correct
// increments of the lsp, to calculate vmisp and to
// protect memory.
//-----
VM_aloc:

sub     lsp, #VM_wordSize     // aloc 1 word
cmp     lsp, spl
movhi   r0, #VM_DUMP_SPBad_aloc
bhi     VM_dumpAbort
mov     vmisp, lsp
add     vmisp, sspace
mov     pc, lr

//-----
// VM_main
//
//-----
.section .text

```

```

.globl VM_main
VM_main:
    // save current registers
PCBaddr    .req r0
push      {r4, r5, r6, r7, r8, r9, r10, r11, r12, lr}
push      {r0} // address current process PCB
add       PCBaddr, #0x08 // PCB_pspace
ldr       pspace, [PCBaddr]
add       PCBaddr, #0x04 // PCB_psl
ldr       psl, [PCBaddr]
add       PCBaddr, #0x04 // PCB_sspace
ldr       sspace, [PCBaddr]
add       PCBaddr, #0x04 // PCB_spl
ldr       spl, [PCBaddr]
add       PCBaddr, #0x04 // PCB_lpc
ldr       lpc, [PCBaddr]
add       PCBaddr, #0x04 // PCB_lsp
ldr       lsp, [PCBaddr]
mov       vmisp, lsp
add       vmisp, sspace // Set to actual memory location
add       PCBaddr, #0x04
ldr       ccntr, [PCBaddr] // PCB_cycles
push      {ccntr}
.unreq    PCBaddr

//-----
// Memory Protection, pspace and fetch next instruction
//-----
VM_loop:
pop       {ccntr}
cmp       ccntr, #0
moveq     r1, #PCB_STATE_ready
beq       VM_main_exit
add       ccntr, #-1
push      {ccntr}
cmp       lpc, psl // ensure the instruction in space
movhs     r0, #VM_DUMP_PSBad // set dump return code.
bhs      VM_dumpAbort // ensut in program space
mov       vmisp, lpc
add       vmisp, pspace // calculate instruction address
ldr       vmisp, [vmisp] // get the instruction

// Dump and abort on bad instruction
cmp       vmisp, #VM_maxInstruct

```

```

blt      VM_main_goodInstruction
mov      r0, #VM_DUMP_InstBad
b        VM_dumpAbort

```

VM_main_goodInstruction:

```

//-----
// Increment PC
//-----
add      lpc, #VM_wordSize // point pc to next instruction
//-----
// push support
//
// stack[--sp] = ps[pc+1]
//
//-----

```

VM_main_psh:

```

cmp      vmdc, #VM_psh // if instruction == push
bne     VM_main_drawCharacter // else try drawCharacter
bl      VM_aloc // point next place in stack
mov     vmpc, lpc // position of the operand
add     vmpc, pspace
ldr     vmdc, [vmpc] // get operand
str     vmdc, [vmsp] // store on top of the stack.
add     lpc, #VM_wordSize // Point to next instruction
b       VM_loop // next instruction

```

```

//-----
// drawCharacter support
//
// Pre-condition Stack:
//      Color
//      x
//      y
//      character
//      --rest--
//
// Post-condition Stack
//      --rest--
//-----

```

VM_main_drawCharacter:

```

cmp      vmdc, #VM_drawCharacter
bne     VM_main_drawString

ldr      r0, [vmsp] // pop color

```

```

bl      VM_dloc
bl      graphicSetFGColor      // Setcolor

ldr     r1, [vmsp]             // pop x pos
bl      VM_dloc

ldr     r2, [vmsp]             // pop y pos
bl      VM_dloc

ldr     r0, [vmsp]             // pop character
bl      VM_dloc

bl      drawCharacter
b       VM_loop                // next instruction
//-----
// drawString support
//
// Pre-condition Stack:
//     Color
//     x
//     y
//     String ([length][text])
//     --rest--
//
// Post-condition Stack
//     --rest--
//-----
VM_main_drawString:

cmp     vmdc, #VM_drawString
bne     VM_main_drawPixel
push   {r4}
ldr     r0, [vmsp]             // pop color
bl      VM_dloc
bl      graphicSetFGColor      // Setcolor
ldr     r1, [vmsp]             // pop x pos
bl      VM_dloc
ldr     r2, [vmsp]             // pop y pos
bl      VM_dloc
mov     r0, vmsp                // get string address
ldr     r4, [vmsp]             // save the string length
bl      drawString
mov     r0, r4
add     r0, #1

```

```

bl      VM_dloc_x
pop     {r4}

b       VM_loop
//-----
// drawPixel support
//// Pre-condition Stack:
//      Color
//      x
//      y
//      --rest--
//
// Post-condition Stack
//      --rest--

//-----
VM_main_drawPixel:
cmp     vmdc, #VM_drawPixel    // if instruction
bne     VM_main_drawLine      // else try Halt
ldr     r0, [vmsp]            // pop color
bl      VM_dloc
bl      graphicSetFGColor     // Setcolor
ldr     r0, [vmsp]            // pop x pos
bl      VM_dloc
ldr     r1, [vmsp]            // pop y pos
bl      VM_dloc
bl      drawPixel
b       VM_loop
//-----
// drawLine support
//// Pre-condition Stack:
//      Color
//      x0
//      y0
//      x1
//      y1
//      --rest--
//
// Post-condition Stack
//      --rest--

//-----
VM_main_drawLine:
cmp     vmdc, #VM_drawLine

```

```

bne      VM_main_fill
ldr      r0, [vmsp]           // pop character
bl       VM_dloc
bl       graphicSetFGColor   // Setcolor
ldr      r0, [vmsp]           // pop x0
bl       VM_dloc
ldr      r1, [vmsp]           // pop y0
bl       VM_dloc
ldr      r2, [vmsp]           // pop x1
bl       VM_dloc
ldr      r3, [vmsp]           // pop y1 ack
bl       VM_dloc
bl       drawLine
b        VM_loop

//-----
// Fill support
// Pre-condition Stack:
//     Color
//     x0
//     y0
//     width
//     height
//     --rest--
//
// Post-condition Stack
//     --rest--
//-----
VM_main_fill:
cmp      vmdc, #VM_fill      // if instruction
bne      VM_main_dup         // else try Halt
ldr      r0, [vmsp]           // pop character
bl       VM_dloc
bl       graphicSetFGColor   // Setcolor
ldr      r0, [vmsp]           // pop x0
bl       VM_dloc
ldr      r1, [vmsp]           // pop y0
bl       VM_dloc
ldr      r2, [vmsp]           // pop width
bl       VM_dloc
ldr      r3, [vmsp]           // pop height
bl       VM_dloc
bl       fillArea
b        VM_loop

```

```

//-----
// dup support
//
// stack[--sp] = stack[sp]
//
//-----
VM_main_dup:
cmp      vmdc, #VM_dup           // if instruction == dup
bne      VM_main_jump          // else try drawCharacter
ldr      vmdc, [vmsp]
bl       VM_aloc
str      vmdc, [vmsp]           // store on top of the stack.
b        VM_loop

//-----
// jmp support
//-----
VM_main_jump:
cmp      vmdc, #VM_jump
bne      VM_main_jmpeq
add      vmpc, #VM_wordSize
ldr      lpc, [vmvc]
b        VM_loop

//-----
// jmq support
// Pre-condition Stack:
//      op0
//      op1
//      --rest--
//
// Post-condition Stack
//      --rest--
//
// Note: two-word instruction
//-----
VM_main_jmpeq:
cmp      vmdc, #VM_jmpeq
bne      VM_main_jmpgte
ldr      vmdc, [vmsp]
bl       VM_dloc
ldr      r0, [vmsp]
bl       VM_dloc
add      vmvc, #VM_wordSize

```

```

add    lpc, #VM_wordSize
cmp    vmdc, r0
ldreq  lpc, [vmvc]
b      VM_loop

//-----
// jmpgte support
// Pre-condition Stack:
//     op0
//     op1
//     --rest--
//
// Post-condition Stack
//     --rest--
// Note: two-word instruction
//-----
VM_main_jmpgte:
cmp    vmdc, #VM_jmpgte
bne    VM_main_jmpgt
ldr    vmvc, [vmvc]
bl     VM_dloc
ldr    r0, [vmvc]
bl     VM_dloc
add    vmvc, #VM_wordSize
add    lpc, #VM_wordSize
cmp    r0, vmvc
ldrge  lpc, [vmvc]
b      VM_loop

//-----
// jmpgt support
// Pre-condition Stack:
//     op0
//     op1
//     --rest--
//
// Post-condition Stack
//     --rest--
// Note: two-word instruction
//-----
VM_main_jmpgt:
cmp    vmvc, #VM_jmpeq
bne    VM_main_jmpneq
/* Reserved for Future Expansion*/
b      VM_loop

```

```

//-----
// jmpneq support
// Pre-condition Stack:
//     op0
//     op1
//     @tgt
//     --rest--
//
// Post-condition Stack
//     --rest--
// Note: two-word instruction
//-----
VM_main_jmpneq:
cmp     vmdc, #VM_jmpneq
bne     VM_main_call
/* Reserved for Future Expansion*/
b       VM_loop
//-----
// Call support
//-----
VM_main_call:
cmp     vmdc, #VM_call
bne     VM_main_ret
add     vmpr, #VM_wordSize
mov     r0, lpc
add     r0, #VM_wordSize
bl      VM_aloc
str     r0, [vmpr]
ldr     lpc, [vmpr]
b       VM_loop
//-----
// ret support
//-----
VM_main_ret:
cmp     vmdc, #VM_ret
bne     VM_main_pop
ldr     lpc, [vmpr]
bl      VM_dloc
b       VM_loop
//-----
// pop support
//-----
VM_main_pop:

```

```

cmp      vmdc, #VM_pop
bne      VM_main_dec
bl       VM_dloc
b        VM_loop

//-----
// dec support
//-----
VM_main_dec:
cmp      vmdc, #VM_dec
bne      VM_main_ror
ldr      vmdc, [vmsp]
sub      vmdc, #1
str      vmdc, [vmsp]
b        VM_loop
//-----
// ror support : currently a NOP
//-----
VM_main_ror:
cmp      vmdc, #VM_ror
bne      VM_main_inc
ldr      vmdc, [vmsp]          // number of places to shift
bl       VM_dloc
ldr      r0, [vmsp]          // value to shift
ror      r0, vmdc
str      r0, [vmsp]
b        VM_loop
//-----
// inc support
//-----
VM_main_inc:

cmp      vmdc, #VM_inc
bne      VM_main_add
ldr      vmdc, [vmsp]
add      vmdc, #1
str      vmdc, [vmsp]
b        VM_loop

//-----
// add support
//
//-----
VM_main_add:

```

```

cmp      vmdc, #VM_add
bne      VM_main_and
ldr      vmdc, [vmsp]
bl       VM_dloc
ldr      r0, [vmsp]
add      vmdc, r0
str      vmdc, [vmsp]
b        VM_loop

//-----
// and support
//-----
VM_main_and:
cmp      vmdc, #VM_and
bne      VM_main_asl
ldr      vmdc, [vmsp]
bl       VM_dloc
ldr      r0, [vmsp]
and      vmdc, r0
str      vmdc, [vmsp]
b        VM_loop

//-----
// asl support
//-----
VM_main_asl:
cmp      vmdc, #VM_asl
bne      VM_main_lsr
ldr      vmdc, [vmsp]           // number of places to shift
bl       VM_dloc
ldr      r0, [vmsp]           // value to shift
lsl      r0, vmdc
str      r0, [vmsp]
b        VM_loop

//-----
// lsr support
//-----
VM_main_lsr:
cmp      vmdc, #VM_lsr
bne      VM_main_asr
ldr      vmdc, [vmsp]           // number of places to shift
bl       VM_dloc
ldr      r0, [vmsp]           // value to shift

```

```

lsr      r0, vmdc
str      r0, [vmsp]
b        VM_loop

//-----
// asr support
//-----
VM_main_asr:
cmp      vmdc, #VM_asr
bne      VM_main_or
ldr      vmdc, [vmsp]      // number of places to shift
bl       VM_dloc
ldr      r0, [vmsp]      // value to shift
asr      r0, vmdc
str      r0, [vmsp]
b        VM_loop
//-----
// or support
//-----
VM_main_or:
cmp      vmdc, #VM_or
bne      VM_main_xor
ldr      vmdc, [vmsp]
bl       VM_dloc
ldr      r0, [vmsp]
orr      vmdc, r0
str      vmdc, [vmsp]
b        VM_loop
//-----
// xor support
//-----
VM_main_xor:
cmp      vmdc, #VM_xor
bne      VM_main_gpioread
ldr      vmdc, [vmsp]
bl       VM_dloc
ldr      r0, [vmsp]
eor      vmdc, r0
str      vmdc, [vmsp]
b        VM_loop
//-----
// GPIO Read support
//
// Pre-condition Stack:

```

```

//          @ read address
//          function constant
//          --rest--
//
// Post-condition Stack
//          vale
//
// Note: Predictiably results can occur if the
//          pin was not set-up via function select.
//
//-----
VM_main_gpioread:
cmp          vmdc, #VM_gpioRead
bne          VM_main_gpioreadset
ldr          r0, [vmsp]
bl           VM_dloc
ldr          r1, [vmsp]
bl           GPIOPinRead
str          r0, [vmsp]
b           VM_loop
//-----
//  GPIO Read Set-upsupport
//
// Pre-condition Stack:
//          @ Pin Fun Sel
//          function constant
//          --rest--
//
// Post-condition Stack
//          vale
//
// Note: Predictiably results can occur if the
//          pin was not set-up via function select.
//
//-----
VM_main_gpioreadset:
cmp          vmdc, #VM_gpioReadSet
bne          VM_main_gpioWrite
ldr          r0, [vmsp]
bl           VM_dloc
ldr          r1, [vmsp]
bl           setGPIOPinFunc
str          r0, [vmsp]
b           VM_loop

```

```

//-----
//  GPIO Write on
//
// Pre-condition Stack:
//      @ read address
//      function constant
//      --rest--
//
// Post-condition Stack
//      vale
//
// Note: Predictably results can occur if the
//      pin was not set-up via function select.
//
//-----
VM_main_gpioWrite:
cmp      vmdc, #VM_gpioWrite
bne      VM_main_swap
ldr      r0, [vmsp]
bl       VM_dloc
ldr      r1, [vmsp]
bl       setGPIOPinFunc
b        VM_loop
//-----
//  swap support
//
// Note: Because swap does not ultimately change the sp
// and because dloc calls are expensive, they are not
// used.  However Memory management maintains one word of
// reserved memory after each sspace to eliminate
// corruption.  Other instructions work similarly.
//-----
VM_main_swap:
cmp      vmdc, #VM_swap
bne      VM_main_sub
ldr      r0, [vmsp]
add      r1, vmsp, #4
ldr      r2, [r1]
str      r2, [vmsp]
str      r0, [r1]
b        VM_loop

//-----
//  sub  support

```

```

//-----
VM_main_sub:
cmp      vmdc, #VM_sub
bne      VM_main_div
ldr      vmdc, [vmsp]    // pop top operand (subtrahend)
bl       VM_dloc
ldr      r0, [vmsp]      // peek second operand (minuend)
sub      r0, vmdc        // determine difference
str      r0, [vmsp]      // replace minuend with difference
b        VM_loop

//-----
// div support
//-----
VM_main_div:
cmp      vmdc, #VM_div
bne      VM_main_mod
ldr      r1, [vmsp]      // pop top operand (divisor)
bl       VM_dloc
ldr      r0, [vmsp]      // peek second operand (quotient)
bl       div             // divide
str      r0, [vmsp]
b        VM_loop

//-----
// mod support
//-----
VM_main_mod:
cmp      vmdc, #VM_mod
bne      VM_main_mul
ldr      r1, [vmsp]      // pop top operand (divisor)
bl       VM_dloc
ldr      r0, [vmsp]      // peek (quotient)
bl       mod             // divide
str      r0, [vmsp]
b        VM_loop

//-----
// mul support
//-----
VM_main_mul:
cmp      vmdc, #VM_mul
bne      VM_main_not
ldr      r1, [vmsp]      // get top operand
(multiplicand)

```

```

bl      VM_dloc
ldr     r0, [vmsp]           // get second operand
(multiplier)
mul     r0, r1               // multiply
str     r0, [vmsp]          // save product
b       VM_loop
//-----
// not support
//-----
VM_main_not:
cmp     vmdc, #VM_not
bne     VM_main_neg
ldr     vmdc, [vmsp]
mvn     r0, vmdc
str     r0, [vmsp]
b       VM_loop
//-----
// neg support
//-----
VM_main_neg:
cmp     vmdc, #VM_neg
bne     VM_main_pause
ldr     vmdc, [vmsp]
mvn     r0, vmdc
add     r0, #1
str     r0, [vmsp]
b       VM_loop
//-----
// pause support
//-----
VM_main_pause:
cmp     vmdc, #VM_pause
bne     VM_main_msg_open
pop     {ccntr}
mov     ccntr, #0
push   {ccntr}
b       VM_loop
//-----
// msgopen support
// Pre-condition Stack:
//     Buffer Size
//     PID of receiving buffer
//     --rest--
//

```

```

// Post-condition Stack
//      Open Status {(0, open), (!0, failed open)}
//      --rest--
//-----
VM_main_msg_open:
cmp      vmdc, #VM_msgOpen
bne      VM_main_msg_write
buffSize .req      r4
pid      .req      r5
push {r4, r5}
ldr      buffSize, [vmsp]
bl       VM_dloc
ldr      pid, [vmsp]
bl       VM_dloc
        // Free check {(r0, pid)}
mov      r0, pid
bl       msgOpenChk
cmp      r0, #0
beq      VM_chk_MO_Lock
bl       VM_aloc
str      r0, [vmsp]
b        VM_loop

        // If Free check = 0, check lock
        // -- checklck{(r0, &SenderPCB)}
VM_chk_MO_Lock:
tgtPtr   .req      r6
push     {r6}
mov      r6, r1
ldr      r0, =schedReadyCurr
bl       msgOpenChkLock
        //if locked
cmp      r1, #0
beq      VM_notLocked
bl       VM_aloc
str      r1, [vmsp]
pop      {r6}
mov      pc, lr
VM_notLocked:
        // If lock ready, lock
        // -- lock      {(r0, &SenderPCB), (r1, BufferSize)}
ldr      r0, =schedReadyCurr

mov      r1, buffSize

```

```

bl      msgOpenLock

      // Complete buffer {(r0, &SenderPCB), (r1, &receiverPCB)}
ldr     r0, =schedReadyCurr
mov     r1, tgtPtr

bl      msgOpenDone
pop     {r6}
pop     {r4, r5}
b       VM_loop
//-----
// msg write support
//-----
VM_main_msg_write:
cmp     vmdc, #VM_msgWrite
bne     VM_main_msg_read
mov     r0, sspace
ldr     r1, [vmsp]
bl      VM_dloc
mov     r2, lsp
bl      msgWrite
      // There is a trick to this code.
      // The size of the buffer is one space
      // above the top of the sspace stack
bl      VM_aloc
ldr     r0, [vmsp]
bl      VM_dloc_x
b       VM_loop
//-----
// msg read support
//
//-----
VM_main_msg_read:
cmp     vmdc, #VM_msgRead
bne     VM_main_time
mov     r0, sspace
bl      msgRead
cmp     r0, #0
ble     VM_loop
bl      VM_dloc_x
b       VM_loop
//-----
// time support
//-----

```

VM_main_time:

```

cmp      vmdc, #VM_time
bne      VM_main_bin2str
bl       getTimeStamp
bl       VM_alloc
str      r0, [vmsp]
bl       VM_alloc
str      r1, [vmsp]
b        VM_loop
//-----
// bin2str support
//
//-----

```

VM_main_bin2str:

```

isNeg    .req r4           // negative number
a        .req r0           // number in process
n        .req r1           // number to divide
temp     .req r2           // temp
length   .req      r5      // lengtht of the string

cmp      vmdc, #VM_bin2str
bne      VM_main_halt
push     {r4, r5}
ldr      a, [vmsp]        // pop value to convert
bl       VM_dloc
mov      isNeg, #0
cmp      a, #0            // if not negative
bge      VM_main_bin2str_notneg // -- Continue
// else
mov      isNeg, #1        // -- Set negative flag
mvn     a, a              // -- take two's complement
add     a, #1

```

VM_main_bin2str_notneg:

```

mov      length, #0       // counter for string length
mov      n, #10          // set divisor;

```

VM_main_bin2str_loop:

```

mov      temp, a         // value of a for div.
// PRE: r0 = a, r1=n=10
bl       mod
// Post r0 = r,

```

```

add    r0, #0x00000030    // convert digit to char
bl     VM_alloc
str    r0, [vmsp]
add    length, #1        // inc length of string

mov    r0, temp          // restore a
                        // PRE: r0 = a, r1=n=10
bl     div
                        // POST: r0 = q (the new a),
cmp    a, #0            // is number is zero
bgt    VM_main_bin2str_loop
cmp    isNeg, #1        // If negative push the -
sign
bne    VM_main_bin2str_loop_exit
mov    r0, #0x0000002D    // neg sign
bl     VM_alloc
str    r0, [vmsp]
add    length, #1

```

VM_main_bin2str_loop_exit:

```

bl     VM_alloc
str    length, [vmsp]    // length
pop    {r4,r5}
b      VM_loop

```

```

.unreq isNeg
.unreq a
.unreq n
.unreq temp
.unreq length

```

```
//-----
```

```
// halt support
```

```
//-----
```

VM_main_halt:

```

cmp    vmdc, #VM_halt
bne    VM_main_dump

```

```

pop    {ccntr}
mov    r1, #PCB_STATE_terminated
b      VM_main_exit

```

```
//-----
```

```
// dump support
```

```
//-----
```

VM_main_dump:

```

cmp      vmdc, #VM_dump
bne      VM_main_nop
mov      r0, #VM_DUMP_USR
b        VM_dumpAbort

```

```

//-----
// nop support
//-----

```

```

ldr      r0, =VM_Dump_On_NOP
cmp      r0, #1
bne      VM_loop
mov      r0, #VM_DUMP_NOP
bl      VM_dumpAbort
b        VM_loop

```

VM_main_exit:

```

// restore registers
PCBaddr  .req r0
pop      {r0} // restore &current PCB
cmp      r1, #PCB_STATE_terminated
movne   r1, #PCB_STATE_ready
add     PCBaddr, #4 // PCB_state
str     r1, [PCBaddr]
add     PCBaddr, #0x00000014 // PCB_lpc
str     lpc, [PCBaddr]
add     PCBaddr, #4 // PCB_lsp
str     lsp, [PCBaddr]
pop     {r4, r5, r6, r7, r8, r9, r10, r11, r12, lr}
mov     pc, lr

```

```

//-----
// VM_dump
//
// Registers
//
//      r0      : Dump code
//-----

```

VM_dumpAbort:

```

mov      r1, #2
mov      r2, #10
bl      drawCharacter

```

```

mov     r0, #0x0000003A
mov     r1, #20
mov     r2, #10
bl      drawCharacter

```

```

mov     r0, #0x00000044
mov     r1, #28
mov     r2, #10
bl      drawCharacter

```

```

mov     r0, #0x00000075
mov     r1, #36
mov     r2, #10
bl      drawCharacter

```

```

mov     r0, #0x0000006D
mov     r1, #42
mov     r2, #10
bl      drawCharacter

```

```

mov     r0, #0x00000070
mov     r1, #50
mov     r2, #10
bl      drawCharacter

```

```

// stack dump

```

```

char    .req r0
x       .req r1
y       .req r2
next    .req r3
stopAddr .req r4

```

```

push    {r4}
mov     next, sspace           // compute first char to draw
mov     stopAddr, sspace
add     stopAddr, spl         // compute last address to draw
mov     x, #12                // Start at ULHC of dump area
mov     y, #0x000000F8

```

```

VM_dump_loop:

```

```

ldr     char, [next]
add     x, #8

```

```

push    {x,y,next}
bl      drawCharacter

pop     {x,y,next}
add     next, #VM_wordSize
cmp     next, stopAddr
ble     VM_dump_loop
beq     VM_dump_loop
pop     {r4}

pop     {ccntr}
mov     r1, #PCB_STATE_terminated
b       VM_main_exit

.unreq  vmpc
.unreq  vmsp
.unreq  vmdc
.unreq  pspace
.unreq  sspace
.unreq  lpc
.unreq  lsp
.unreq  psl
.unreq  spl
.unreq  ccntr

```

9.2 Customizer Source Code

This section includes the source code for the Java Build Customizer tool.

9.2.1 ConstTable.java

```

package jjp.reos.reosvm;

import java.awt.Color;
/**
 *
 * @author James J. Perry
 * This table is used to resolve color.
 *

```

```

*/
public class ConstTable {
    static String[] symbol = {"Color.red",
        "Color.blue", "Color.green",
        "Color.black", "Color.white"};
    static String[] xlate = {"0x0000F800",
        "0x000000FF", "0x000007E0",
        "0x00000000", "0x000FFFFFF"};

    public ConstTable() {

    }

    public static String getConst (String s){
        System.out.println("|"+s+"|");
        for(int x = 0; x<symbol.length; x++){
            if (symbol[x].equals(s)){
                return xlate[x];
            }
        }
        return "";
    }
}

```

9.2.2 CustConfig.java

```

package jjp.reos.reosvm;

/**
 * Custom configuration code. This module converts
 * the user's input to the customizer tool and
 * encapsulates them into an objectthat is used as
 * the * input to the build process.
 *
 * @author LordHighCommander
 */
public class CustConfig {
    // CPU Memory
    int cpuMem;

    // PVT Refresh
    boolean pvtRefresh;
}

```

```
// GPIO Support
boolean gpioPin;

boolean gpioOKLED;

// Color Mode
int colorMode;

public static final int BW2Bit=1;
public static final int GR8Bit=2;
public static final int CL3Bit=3;
public static final int CL8Bit=4;
public static final int CL16Bit=5;
public static final int CL24Bit=6;

// Display Mode
// TO DO: add list of supported modes.
int displayMode;

// SD Card Support
int sdCardType;

public static final int SD4 = 0;
public static final int SD8 = 1;
public static final int SD16 = 2;
public static final int SD32 = 3;

// Process Execution--scheduling mode
boolean schedMode;

public static final boolean SCH_STATE_spe=false;
public static final boolean SCH_STATE_mpce=true;

// Process Execution--queuing mode
boolean schedQueueType;

public static final boolean SCH_STATE_queue=true;
public static final boolean SCH_STATE_stack=false;
int quantum;

// Virtual Memory Support
boolean virtualMemory;

// File I/O
```

```
boolean fileIO;

// Inter-process Messaging Support

boolean processMsg;

// Software Support

boolean drawing;
boolean string;

// Fail/DumpSupport

boolean dumpOnUnsupported;
boolean failOnUserDump;

// Memory Allocation Support

boolean memAllo;
public static final boolean firstFit=true;
public static final boolean bestFit=false;

public CustConfig(int mem, boolean pvt, boolean gpp,
                 boolean gpok, int cm, int dm, int sd,
                 boolean sm, boolean sq, int q, boolean vm,
                 boolean fio, boolean pm, boolean dr,
                 boolean st, boolean dumpUp, boolean foud,
                 boolean mal){

    // Memory
    cpuMem=mem;

    // PVT Refresh
    pvtRefresh = pvt;

    // GPIO Support
    gpioPin=gpp;
    gpioOKLED=gpok;

    // Color Mode
    colorMode=cm;

    // Display Mode
    displayMode=dm;
```

```

// SD Card Support
sdCardType =sd;

// Process Execution--scheduling mode
schedMode = sm;

// Process Execution--queuing mode
schedQueueType = sq;

// Process Quantum

quantum = q;

// Virtual Memory Support
virtualMemory =vm;

// File I/O
fileIO=fio;

// Inter-process Messaging Support

processMsg=pm;

// Software Support

drawing = dr;
string = st;

// Fail/DumpSupport

dumpOnUnsupported = dumpUp;
failOnUserDump = foud;

memAllo=mal;
}

public String toString(){
    return "{\n"
        +"(cpuMem, "+cpuMem+"),\n"
        +"(pvtRefresh, "+pvtRefresh+"),\n"
        +"(gpioPin, "+gpioPin+"),\n"
        +"(gpioOKLED, "+gpioOKLED+"),\n"
        +"(colorMode, "+colorMode+"),\n"

```

```

        +(displayMode, "+displayMode+"),\n"
        +(sdCardType, "+sdCardType+"),\n"
        +(schedMode, "+schedMode+"),\n"
        +(schedQueueType, "+schedQueueType+"),\n"
        +(virtualMemory, "+virtualMemory+"),\n"
        +(fileIO, "+fileIO+"),\n"
        +(processMsg, "+processMsg+"),\n"
        +(drawing, "+drawing+"),\n"
        +(string, "+string+"),\n"
        +(failOnUnsupported,
+"dumpOnUnsupported+"),\n"
        +(failOnUserDump, "+failOnUserDump+"),\n"
        +(memAllo, "+memAllo+"),\n"
        +"}\n";
    }
}

```

9.2.3 GPIOConst.java

```

package jjp.reos.reosvm;
/**
 * Function selection words for GPIO
 * @author LordHighCommander
 *
 */
public class GPIOConst {

//-----
//  Function Sel address for GPIO 0 to 31 is 0x20200000.
//  Function Sel address for GPIO 32 to 53 is 020200004.
//-----

    public static final String [] GPIO_FuncOut =
    {
        "0x00000001", "0x00000004", "0x00000040",
        "0x00000200", "0x00001000", "0x00040000",
        "0x00200000", "0x01000000", "0x04000000",
        "0x20000000", "0x00000001", "0x00000004",
        "0x00000040", "0x00000200", "0x00001000",
        "0x00040000", "0x00200000", "0x01000000",
        "0x04000000", "0x20000000", "0x00000001",

```

```

        "0x00000004", "0x00000040", "0x00000200",
        "0x00001000", "0x00040000", "0x00200000",
        "0x01000000", "0x04000000", "0x20000000",
        "0x00000001", "0x00000004", "0x00000040",
        "0x00000200", "0x00001000", "0x00040000",
        "0x00200000", "0x01000000", "0x04000000",
        "0x20000000", "0x00000001", "0x00000004",
        "0x00000040", "0x00000200", "0x00001000",
        "0x00040000", "0x00200000", "0x01000000",
        "0x04000000", "0x20000000", "0x00000001",
        "0x00000004", "0x00000040", "0x00000200",
        "0x00001000", "0x00040000", "0x00200000",
        "0x01000000", "0x04000000", "0x20000000",
        "0x00000001", "0x00000004", "0x00000040",
        "0x00000200"
    };

    public static String getGPIO_FuncSelAddr(int n){
        if (n<32) return "0x20200000";
        else return "0x20200004";
    }
}

```

9.2.4 REOSCustomizer.java

```

package jjp.reos.reosvm;

import javax.swing.ButtonGroup;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JRadioButton;
import javax.swing.JTextArea;
import javax.swing.JTextField;
import java.awt.event.*;
import java.io.File;
import java.util.ArrayList;

@SuppressWarnings("serial")
public class REOSCustomizer extends JFrame {

    final static int frameWidth = 1024;
    final static int frameHeight = 720;
    final static int fieldWidth = 231;

```

```

final static int column1 = 5;
final static int column2 = 256;
final static int column3 = 512;
final static int column4 = 768;
final static int headerHeight = 40;
REOSJVMassm assm = new REOSJVMassm();
JLabel status;
JLabel nextLbl;

ArrayList<String> residentList
    = new ArrayList<String>();

public REOSCustomizer(){

    File[] file
        = new File("./rEOSBuild/").listFiles();
    for (File f : file){
        f.delete();
    }
    createFrame();
    createInterface();
    this.setVisible(true);
}

private void createFrame(){
    this.setSize (frameWidth, frameHeight);
    this.setTitle("rEOS Assembler, Customization"
        +"and Build Dashboard v0.1");
    this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    this.setLayout(null);
}

private void createInterface(){
    status = new JLabel ();
    status.setBounds(column1, 10, frameWidth-20, 20);
    this.add(status);

    hardwareInterface();
    operatingSystemInterface();
    assemblyInterface();
    statusInterface();
}

JButton memorySuggestionButton;

```

```

JTextField memorySetting;
JRadioButton disablePvt;
JRadioButton gpioSupport;
JRadioButton gpioOKSupport;
ButtonGroup colorType;
JRadioButton mono2Support;
JRadioButton gray8Support;
JRadioButton color3Support;
JRadioButton color8Support;
JRadioButton color16Support;
JRadioButton color24Support;
private static final int mono2=1;
private static final int gray8=2;
private static final int color3=3;
private static final int color8=4;
private static final int color16=5;
private static final int color24=6;
JLabel display;
JLabel displayChoice;
JButton displayButton;
ButtonGroup sdCardType;
JRadioButton sdCard4Support;
JRadioButton sdCard8Support;
JRadioButton sdCard16Support;
JRadioButton sdCard32Support;
private static final int sdCard4=1;
private static final int sdCard8=2;
private static final int sdCard16=3;
private static final int sdCard32=4;

private void hardwareInterface(){
    nextLbl = new JLabel("Hardware Configuration");
    nextLbl.setBounds(column1,
                      headerHeight+10,
                      fieldWidth, 20);
    this.add(nextLbl);

    nextLbl
        = new JLabel("CPU/GPU Split; CPU Memory
(kbytes):");

    nextLbl.setBounds(column1,
                      headerHeight+40,
                      fieldWidth, 20);

```

```

this.add(nextLbl);

memorySetting = new JTextField("128");
memorySetting.setBounds(column1,
                        headerHeight+60,
                        fieldWidth, 20);
memorySetting.setEditable(true);
this.add(memorySetting);

memorySuggestionButton
    = new JButton("GET MEMORY SUGGESTION");
memorySuggestionButton.setBounds(column1,
                                headerHeight+85,
                                fieldWidth, 40);
memorySuggestionButton.setEnabled(false);
memorySuggestionButton.addActionListener(
    new ActionListener(){
        public void actionPerformed(ActionEvent e){

            }
    }
);
this.add(memorySuggestionButton);

nextLbl = new JLabel("PVT Refresh:");
nextLbl.setBounds(column1,
                 headerHeight+140,
                 fieldWidth, 20);
this.add(nextLbl);

disablePvt = new JRadioButton("Disable Refresh");
disablePvt.setSelected(false);
disablePvt.setBounds(column1,
                    headerHeight+160,
                    fieldWidth, 20);
disablePvt.setEnabled(true);
this.add(disablePvt);

nextLbl = new JLabel("GPIO Support:");
nextLbl.setBounds(column1,
                 headerHeight+190,
                 fieldWidth, 20);
this.add(nextLbl);

```

```
gpioSupport = new JRadioButton("GPIO PIN Support");
gpioSupport.setSelected(false);
gpioSupport.setBounds(column1,
    headerHeight+210,
    fieldWidth, 20);
gpioSupport.setEnabled(true);
this.add(gpioSupport);

gpioOKSupport = new JRadioButton("GPIO OK Support");
gpioOKSupport.setSelected(false);
gpioOKSupport.setBounds(column1,
    headerHeight+230,
    fieldWidth, 20);
gpioOKSupport.setEnabled(true);
this.add(gpioOKSupport);

nextLbl = new JLabel("Color Mode:");
nextLbl.setBounds(column1,
    headerHeight+260,
    fieldWidth, 20);
this.add(nextLbl);

colorType = new ButtonGroup();

mono2Support = new JRadioButton
    ("2 Bit Black and White");
mono2Support.setSelected(false);
mono2Support.setBounds(column1,
    headerHeight+280,
    fieldWidth, 20);
mono2Support.setEnabled(false);
this.add(mono2Support);
colorType.add(mono2Support);

gray8Support
    = new JRadioButton("8 Bit Grayscale");
gray8Support.setSelected(false);
gray8Support.setBounds(column1,
    headerHeight+300,
    fieldWidth, 20);
gray8Support.setEnabled(false);
this.add(gray8Support);
colorType.add(gray8Support);
```

```
color3Support
    = new JRadioButton("3 Bit Color");
color3Support.setSelected(false);
color3Support.setBounds(column1,
    headerHeight+320,
    fieldWidth, 20);
color3Support.setEnabled(false);
this.add(color3Support);
colorType.add(color3Support);

color8Support
    = new JRadioButton("8 Bit Low Color");
color8Support.setSelected(false);
color8Support.setBounds(column1,
    headerHeight+340,
    fieldWidth, 20);
color8Support.setEnabled(true);
this.add(color8Support);
colorType.add(color8Support);

color16Support
    = new JRadioButton("16 Bit High Color");
color16Support.setSelected(true);
color16Support.setBounds(column1,
    headerHeight+360,
    fieldWidth, 20);
color16Support.setEnabled(true);
this.add(color16Support);
colorType.add(color16Support);

color24Support
    = new JRadioButton("24 Bit True Color");
color24Support.setSelected(false);
color24Support.setBounds(column1,
    headerHeight+380,
    fieldWidth, 20);
color24Support.setEnabled(true);
this.add(color24Support);
colorType.add(color24Support);

nextLbl = new JLabel("Display Customization:");
nextLbl.setBounds(column1,
    headerHeight+410,
    fieldWidth, 20);
```

```

this.add(nextLbl);

displayChoice
    = new JLabel((char)0x25ef+" Default HDMI");
displayChoice.setBounds(column1,
                        headerHeight+430,
                        fieldWidth, 20);
this.add(displayChoice);

displayButton
    = new JButton("DISPLAY SETTINGS");
displayButton.setBounds(column1,
                       headerHeight+450,
                       fieldWidth,
                       40);
displayButton.setEnabled(false);
displayButton.addActionListener(
    new ActionListener() {
        public void actionPerformed(ActionEvent e) {

            }
    }
);
this.add(displayButton);

nextLbl = new JLabel("SD Card Support:");
nextLbl.setBounds(column1,
                 headerHeight+505,
                 fieldWidth,
                 20);
this.add(nextLbl);

sdCardType = new ButtonGroup();

sdCard4Support
    = new JRadioButton("4 GB SD Card");
sdCard4Support.setSelected(false);
sdCard4Support.setBounds(column1,
                        headerHeight+525,
                        fieldWidth,
                        20);
sdCard4Support.setEnabled(false);
this.add(sdCard4Support);
sdCardType.add(sdCard4Support);

```

```

sdCard8Support
    = new JRadioButton("8 GB SD Card");
sdCard8Support.setSelected(false);
sdCard8Support.setBounds(column1,
    headerHeight+545,
    fieldWidth,
    20);
sdCard8Support.setEnabled(false);
this.add(sdCard8Support);
sdCardType.add(sdCard8Support);

sdCard16Support
    = new JRadioButton("16 GB SD Card");
sdCard16Support.setSelected(true);
sdCard16Support.setBounds(column1,
    headerHeight+565,
    fieldWidth,
    20);
sdCard16Support.setEnabled(false);
this.add(sdCard16Support);
sdCardType.add(sdCard16Support);

sdCard32Support
    = new JRadioButton("32 GB SD Card");
sdCard32Support.setSelected(false);
sdCard32Support.setBounds(column1,
    headerHeight+585,
    fieldWidth,
    20);
sdCard32Support.setEnabled(false);
this.add(sdCard32Support);
sdCardType.add(sdCard32Support);
}

ButtonGroup schedType;
JRadioButton sPEStackButton;
JRadioButton sPEQueueButton;
JRadioButton mPCEButton;
JTextField quantum;
JRadioButton virtualMemoryButton;
JRadioButton fileIOButton;
JRadioButton msgButton;

```

```

JRadioButton drawButton;
JRadioButton stringButton;
ButtonGroup dumpType;
JRadioButton failOnUnknown;
JRadioButton failOnDump;
JRadioButton bestFit;
JRadioButton firstFit;
ButtonGroup memoryMgtType;

private void operatingSystemInterface() {
    nextLbl
        = new JLabel("Operating System Configuration");
    nextLbl.setBounds(column2,
                     headerHeight+10,
                     fieldWidth,
                     20);
    this.add(nextLbl);

    nextLbl
        = new JLabel("Process Execution Mode:");
    nextLbl.setBounds(column2,
                     headerHeight+40,
                     fieldWidth,
                     20);
    this.add(nextLbl);

    schedType = new ButtonGroup();

    sPEStackButton
        = new JRadioButton("SPE-Stack Mode");
    sPEStackButton.setSelected(false);
    sPEStackButton.setBounds(column2,
                             headerHeight+60,
                             fieldWidth,
                             20);
    sPEStackButton.setEnabled(true);
    this.add(sPEStackButton);
    schedType.add(sPEStackButton);

    sPEQueueButton
        = new JRadioButton("SPE-Queue Mode");
    sPEQueueButton.setSelected(true);
    sPEQueueButton.setBounds(column2,
                             headerHeight+80,

```

```

        fieldWidth,
        20);
sPEQueueButton.setEnabled(true);
this.add(sPEQueueButton);
schedType.add(sPEQueueButton);

mPCEButton
    = new JRadioButton("MPCE Mode");
mPCEButton.setSelected(false);
mPCEButton.setBounds(column2,
    headerHeight+100,
    fieldWidth,
    20);
mPCEButton.setEnabled(true);
this.add(mPCEButton);
schedType.add(mPCEButton);

nextLbl
    = new JLabel("Round Robin Quantum:");
nextLbl.setBounds(column2+30,
    headerHeight+120,
    fieldWidth-30,
    20);
this.add(nextLbl);

quantum = new JTextField("5");
quantum.setBounds(column2+30,
    headerHeight+140,
    fieldWidth-30,
    20);
quantum.setEditable(true);
this.add(quantum);

nextLbl
    = new JLabel("Virtual Memory Support:");
nextLbl.setBounds(column2,
    headerHeight+170,
    fieldWidth,
    20);
this.add(nextLbl);

virtualMemoryButton
    = new JRadioButton("Virtual Memory Support");
virtualMemoryButton.setSelected(false);

```

```
virtualMemoryButton.setBounds(column2,
    headerHeight+190,
    fieldWidth,
    20);
virtualMemoryButton.setEnabled(false);
this.add(virtualMemoryButton);

nextLbl = new JLabel("File Input/Output Support:");
nextLbl.setBounds(column2,
    headerHeight+220,
    fieldWidth,
    20);
this.add(nextLbl);

fileIOButton
    = new JRadioButton("File I/O Support");
fileIOButton.setSelected(false);
fileIOButton.setBounds(column2,
    headerHeight+240,
    fieldWidth, 20);
fileIOButton.setEnabled(false);
this.add(fileIOButton);

nextLbl
    = new JLabel("Inter-process Messaging Support:");
nextLbl.setBounds(column2,
    headerHeight+270,
    fieldWidth,
    20);
this.add(nextLbl);

msgButton = new JRadioButton("Messaging Support");
msgButton.setSelected(false);
msgButton.setBounds(column2,
    headerHeight+290,
    fieldWidth,
    20);
msgButton.setEnabled(true);
this.add(msgButton);

nextLbl = new JLabel("OS Software Support:");
nextLbl.setBounds(column2,
    headerHeight+320,
    fieldWidth,
```

```
        20);
this.add(nextLbl);

drawButton = new JRadioButton("Drawing Support");
drawButton.setSelected(false);
drawButton.setBounds(column2,
    headerHeight+340,
    fieldWidth,
    20);
drawButton.setEnabled(true);

this.add(drawButton);

stringButton = new JRadioButton("String Support");
stringButton.setSelected(false);
stringButton.setBounds(column2,
    headerHeight+360,
    fieldWidth,
    20);
stringButton.setEnabled(true);
this.add(stringButton);

nextLbl = new JLabel("Fail/Dump Support:");
nextLbl.setBounds(column2,
    headerHeight+390,
    fieldWidth,
    20);
this.add(nextLbl);

failOnUnknown
    = new JRadioButton("Dump on unsupported"
        +"command");
failOnUnknown.setSelected(true);
failOnUnknown.setBounds(column2,
    headerHeight+410,
    fieldWidth,
    20);
failOnUnknown.setEnabled(true);
this.add(failOnUnknown);

failOnDump
    = new JRadioButton("Fail on user-initiated"
        +"dump");
failOnDump.setSelected(true);
```

```

failOnDump.setBounds(column2,
                      headerHeight+430,
                      fieldWidth,
                      20);
failOnDump.setEnabled(true);
this.add(failOnDump);

nextLbl
    = new JLabel("Memory Allocation Support:");
nextLbl.setBounds(column2,
                  headerHeight+460,
                  fieldWidth,
                  20);
this.add(nextLbl);

memoryMgtType = new ButtonGroup();

firstFit = new JRadioButton("First Fit");
firstFit.setSelected(true);
firstFit.setBounds(column2,
                   headerHeight+480,
                   fieldWidth,
                   20);
firstFit.setEnabled(true);
this.add(firstFit);
memoryMgtType.add(firstFit);

bestFit = new JRadioButton("Best Fit");
bestFit.setSelected(false);
bestFit.setBounds(column2,
                  headerHeight+500,
                  fieldWidth,
                  20);
bestFit.setEnabled(true);
this.add(bestFit);
memoryMgtType.add(bestFit);
}

JTextField pathname;
JTextField filename;
JButton assembleButton;
JLabel pidLbl;
JTextField pid;
JRadioButton resident;

```



```

pid = new JTextField ("1");
pid.setBounds(column3,
              headerHeight+130,
              fieldWidth,
              20);
pid.setEditable(true);
this.add(pid);

sSpaceSizeLbl
    = new JLabel ("Data Space Requested (in
bytes):");
sSpaceSizeLbl.setBounds(column3,
                        headerHeight+160,
                        fieldWidth,
                        20);
this.add(sSpaceSizeLbl);

sSpaceSize = new JTextField ("64");
sSpaceSize.setBounds(column3,
                    headerHeight+180,
                    fieldWidth,
                    20);
sSpaceSize.setEditable(true);
this.add(sSpaceSize);

resident
    = new JRadioButton("Make Resident Process");
resident.setSelected(false);
resident.setBounds(column3,
                  headerHeight+210,
                  fieldWidth,
                  20);
this.add(resident);

nextLbl = new JLabel("Generated PCB:");
nextLbl.setBounds(column3,
                 headerHeight+230,
                 fieldWidth,
                 20);
this.add(nextLbl);

pcb = new JTextArea();
pcb.setEditable(false);

```

```

pcb.setText("Not yet generated:\n\n");
pcb.setBounds(column3,
              headerHeight+210,
              fieldWidth,
              220);
this.add(pcb);

assembleButton = new JButton("ASSEMBLE");
assembleButton.setBounds(column3,
                         frameHeight-100,
                         fieldWidth,
                         40);
assembleButton.addActionListener(
    new ActionListener() {
        public synchronized void
actionPerformed(ActionEvent e) {
            int p=0;
            int sSpaceReq;
            try{

sSpaceReq=Integer.parseInt(sSpaceSize.getText());
                p =
Integer.parseInt(pid.getText());
            }
            catch(Exception ex){
                status.setText("Invalid Stack
size");

                return;
            }

pcb.setText(assm.assemble(pathname.getText(),
                          filename.getText(),
                          pid.getText(),
                          sSpaceReq,
                          resident.isSelected()));
            if(assm.getStatus().equals("Success")) {
                boolean add = true;
                for (String s : residentList){

if(s.equals(filename.getText())) {
                    add = false;
                }
            }
            if(add &&resident.isSelected()){

```

```

residentList.add(filename.getText());
                }
            }
            status.setText(assm.getStatus());
        }
    }
);
this.add(assembleButton);
}

```

```

JTextArea buildLbl;
JButton buildButton;

```

```

private synchronized void statusInterface(){
    nextLbl = new JLabel("Build Status");
    nextLbl.setBounds(column4,
                      headerHeight+10,
                      fieldWidth,
                      20);
    this.add(nextLbl);

    nextLbl = new JLabel("Files in the build:");
    nextLbl.setBounds(column4,
                      headerHeight+40,
                      fieldWidth,
                      20);
    this.add(nextLbl);

    buildLbl = new JTextArea();
    buildLbl.setEditable(false);
    buildLbl.setText("vm.s\n");
    buildLbl.setBounds(column4,
                      headerHeight+60,
                      fieldWidth,
                      320);
    this.add(buildLbl);

    buildButton = new JButton("BUILD IMAGE");
    buildButton.setBounds(column4,
                          frameHeight-100,
                          fieldWidth,
                          40);
}

```

```

        buildButton.addActionListener(
            new ActionListener() {
                public synchronized void
actionPerformed(ActionEvent e) {

System.out.println(colorType.getSelection());
                REOSInitFactory r =new
REOSInitFactory((getConfig()),
                    residentList);
                buildLbl.setText(r.makeConfig("./"));

            }
        );
        this.add(buildButton);
    }

    private CustConfig getConfig() {

        int clm=color16;
        if (mono2Support.isSelected())clm = mono2;
        else if (gray8Support.isSelected())clm = gray8;
        else if (color3Support.isSelected())clm = color3;
        else if (color8Support.isSelected())clm = color8;
        else if (color16Support.isSelected())clm = color16;
        else if (color24Support.isSelected())clm = color24;

        int sd=3;

        if (sdCard4Support.isSelected())sd = sdCard4;
        else if (sdCard8Support.isSelected())sd = sdCard8;
        else if (sdCard16Support.isSelected())sd = sdCard16;
        else if (sdCard32Support.isSelected())sd = sdCard32;

        boolean sm=false;
        boolean sq=true;

        if (sPEStackButton.isSelected()) {
            sm=CustConfig.SCH_STATE_spe;
            sq=CustConfig.SCH_STATE_stack;
        }
        else if (sPEQueueButton.isSelected()) {
            sm=CustConfig.SCH_STATE_spe;

```

```

        sq=CustConfig.SCH_STATE_queue;
    }
    else if (mPCEButton.isSelected()) {
        sm=CustConfig.SCH_STATE_mpce;
        sq=CustConfig.SCH_STATE_queue;
    }

    boolean mm = true;
    if (firstFit.isSelected()) mm=CustConfig.firstFit;
    else if (bestFit.isSelected()) mm=CustConfig.bestFit;
    // NOTE?TO_DO:
    // Separate MEnu for DM value needed.
    return new
CustConfig(Integer.parseInt(memorySetting.getText()),
            disablePvt.isSelected(),
            gpioSupport.isSelected(),
            gpioOKSupport.isSelected(),
            clm,
            0,
            sd,
            sm,
            sq,
            Integer.parseInt(quantum.getText()),
            virtualMemoryButton.isSelected(),
            fileIOButton.isSelected(),
            msgButton.isSelected(),
            drawButton.isSelected(),
            stringButton.isSelected(),
            failOnUnknown.isSelected(),
            failOnDump.isSelected(),
            mm);
}

public static void main(String[] args) {

    new REOSCustomizer();

}
}

```

9.2.4 REOSInitFactory

```

package jjp.reos.reosvm;
import java.io.File;
import java.io.FileReader;
import java.io.BufferedReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.nio.file.Files;
import java.nio.file.StandardCopyOption;
import java.util.ArrayList;
/**
 * This class converts the customization data
 * from the Customizer GUI into two build
 * files: init.s, config.txt, and a list of
 * components needed for the build.
 * @author LordHighCommander
 *
 */
public class REOSInitFactory {
    ArrayList<String> initText
        = new ArrayList<String>();
    ArrayList<String> configText
        = new ArrayList<String>();
    ArrayList<String> residentList
        = new ArrayList<String>();

    CustConfig current;

    boolean coreGraphics;
    int disp_x;
    int disp_y;
    boolean drawing;
    boolean fileio;
    boolean gpioPin;
    boolean gpioOKLED;
    boolean mpce;
    boolean processMsg;
    boolean string;
    boolean vid;
    int quantum;
    boolean virtMem;

```

```

boolean dumpUp;

public REOSInitFactory(CustConfig c,
    ArrayList<String> r){
    current = c;
    residentList = r;

}

public String makeConfig(String path){

    if(current.gpioPin){
        gpioPin = true;
    }
    if(current.gpioOKLED){
        gpioOKLED = true;
    }

    if(current.displayMode==0){
        disp_x=1024;
        disp_y=768;
    }
    //+++++
    // ADD COLORMODE AND DISPLAY HERE
    //+++++

    if(current.schedMode){
        mpce = true;
    }

    System.out.println("{(quantum, "+ quantum+"}");
    quantum = Math.max(1, current.quantum);

    if(current.virtualMemory){
        virtMem=true;
    }

    if(current.fileIO){
        fileio=true;
    }
    if(current.processMsg){
        processMsg=true;
    }
}

```

```

    if(current.drawing) {
        coreGraphics=true;
        drawing=true;
        vid=true;
    }

    if(current.string) {
        coreGraphics=true;
        vid=true;
        string=true;
    }

    configText();
    initText();
    return movToBuild();
    //
}

/**
 * Create Config.txt
 */
private void configText() {

configText.add("#####
#####");
    configText.add("# Memory Settings");
    configText.add("# ");
    configText.add("# Note: rEOS configuration tools do
not support CMA Dynamic Memory Split");
    configText.add("#         because it is not officially
supported.");

configText.add("#####
#####");
    configText.add("gpu_mem="+current.cpuMem);
    configText.add("disable_l2cache=0");
    if(current.pvtRefresh) {
        configText.add("disable_pvt=1");
    }
    else{
        configText.add("disable_pvt=0");
    }
}

```

```

configText.add("#####
#####");
    configText.add("# Display Settings");

configText.add("#####
#####");
    switch(current.displayMode){
        case CustConfig.CL8Bit:
            configText.add("framebuffer_depth=8");
        case CustConfig.CL16Bit:
            configText.add("framebuffer_depth=16");
        case CustConfig.CL24Bit:
            configText.add("framebuffer_depth=24");
        default:
    }

configText.add("#####
#####");
    configText.add("# Boot Settings");

configText.add("#####
#####");

    PrintWriter out=null;
    try{
        out = new PrintWriter(new
FileWriter("./rEOSSource/config.txt"));
    }
    catch (IOException ioe){

    }
    for( String s: configText){
        // System.out.println(s);
        out.println(s);
    }

    out.close();
}

/**
 * Move required files to build (always include:
 * config.txt, init.s, main.s, processmgr.s, util.s, vm.s).

```

```

*/
private String movToBuild(){
    String s = "";

    try{
        Files.copy(
            new
File("./rEOSSource/config.txt").toPath(),
            new File("./rEOSBuild/config.txt").toPath(),
            StandardCopyOption.REPLACE_EXISTING);
    }
    catch (Exception e){
        return "Build Failed:\nCopying config.txt.";
    }
    s+="config.txt\n";
    try{
        Files.copy(
            new
File("./rEOSSource/init.s").toPath(),
            new
File("./rEOSBuild/init.s").toPath(),
            StandardCopyOption.REPLACE_EXISTING);
    }
    catch (Exception e){
        return "Build Failed:\nCopying init.s.";
    }
    s+="init.s\n";

    try{
        Files.copy(
            new
File("./rEOSSource/main.s").toPath(),
            new
File("./rEOSBuild/main.s").toPath(),
            StandardCopyOption.REPLACE_EXISTING);
    }
    catch (Exception e){
        return "Build Failed:\nCopying main.s.";
    }
    s+="main.s\n";
    try{
        Files.copy(
            new
File("./rEOSSource/processmgr.s").toPath(),

```

```

        new
File("./rEOSBuild/processmgr.s").toPath(),
        StandardCopyOption.REPLACE_EXISTING);
    }
    catch (Exception e){
        return "Build Failed:\nCopying processmgr.s.";
    }
    s+="processmgr.s\n";
    try{
        Files.copy(
            new
File("./rEOSSource/util.s").toPath(),
            new
File("./rEOSBuild/util.s").toPath(),
            StandardCopyOption.REPLACE_EXISTING);
    }
    catch (Exception e){
        return "Build Failed:\nCopying util.s.";
    }
    s+="util.s\n";
    try{
        Files.copy(
            new File("./rEOSSource/vm.s").toPath(),
            new File("./rEOSBuild/vm.s").toPath(),
            StandardCopyOption.REPLACE_EXISTING);
    }
    catch (Exception e){
        return "Build Failed:\nCopying vm.s.";
    }
    s+="vm.s\n";
    if (coreGraphics){
        s+="coreGraphics.s\n";
        try{
            Files.copy(
                new
File("./rEOSSource/coregraphics.s").toPath(),
                new
File("./rEOSBuild/coregraphics.s").toPath(),
                StandardCopyOption.REPLACE_EXISTING);
        }
        catch (Exception e){
            return "Build Failed:\nCopying
coregraphics.s.";
        }
    }

```

```

    }
    else{
        new File("./rEOSBuild/coreGraphics.s").delete();
    }
    if (drawing){
        s+="drawing.s\n";
        try{
            Files.copy(
                new
File("./rEOSSource/drawing.s").toPath(),
                new
File("./rEOSBuild/drawing.s").toPath(),
                StandardCopyOption.REPLACE_EXISTING);
        }
        catch (Exception e){
            return "Build Failed:\nCopying drawing.s.";
        }
    }
    else{
        new File("./rEOSBuild/drawing.s").delete();
    }
    if (fileio){
        s+="fileio.s\n";
        try{
            Files.copy(
                new
File("./rEOSSource/fileio.s").toPath(),
                new
File("./rEOSBuild/fileio.s").toPath(),
                StandardCopyOption.REPLACE_EXISTING);
        }
        catch (Exception e){
            return "Build Failed:\nCopying fileio.s.";
        }
    }
    else{
        new File("./rEOSBuild/fileio.s").delete();
    }
    if (gpioPin){
        s+="gpiopin.s\n";
        try{
            Files.copy(
                new
File("./rEOSSource/gpiopin.s").toPath(),

```

```

                new
File("./rEOSBuild/gpiopin.s").toPath(),
                StandardCopyOption.REPLACE_EXISTING);
    }
    catch (Exception e){
        return "Build Failed:\nCopying gpiopin.s.";
    }
}
else{
    new File("./rEOSBuild/gpioPin.s").delete();
}
if (gpioOKLED){
    s+="gpiookled.s\n";
    try{
        Files.copy(
            new
File("./rEOSSource/gpiookled.s").toPath(),
            new
File("./rEOSBuild/gpiookled.s").toPath(),
            StandardCopyOption.REPLACE_EXISTING);
    }
    catch (Exception e){
        return "Build Failed:\nCopying
gpiookled.s.";
    }
}
else{
    new File("./rEOSBuild/gpioOKLED.s").delete();
}
if (mpce){
    s+="mpce.s\n";
    try{
        Files.copy(
            new
File("./rEOSSource/mpce.s").toPath(),
            new
File("./rEOSBuild/mpce.s").toPath(),
            StandardCopyOption.REPLACE_EXISTING);
    }
    catch (Exception e){
        return "Build Failed:\nCopying mpce.s.";
    }
}
else{

```

```

        new File("./rEOSBuild/mpce.s").delete();
    }
    if (processMsg) {
        s+="processmsg.s\n";
        try{
            Files.copy(
                new
File("./rEOSSource/processmsg.s").toPath(),
                new
File("./rEOSBuild/processmsg.s").toPath(),
                StandardCopyOption.REPLACE_EXISTING);
        }
        catch (Exception e){
            return "Build Failed:\nCopying
processmsg.s.";
        }
    }
    else{
        new File("./rEOSBuild/processMsg.s").delete();
    }
    if (string){
        s+="string.s\n";
        try{
            Files.copy(
                new
File("./rEOSSource/string.s").toPath(),
                new
File("./rEOSBuild/string.s").toPath(),
                StandardCopyOption.REPLACE_EXISTING);
            Files.copy(
                new
File("./rEOSSource/font.bin").toPath(),
                new
File("./rEOSBuild/font.bin").toPath(),
                StandardCopyOption.REPLACE_EXISTING);
        }
        catch (Exception e){
            return "Build Failed:\n"
+"Copying string.s or font.bin.";
        }
    }
    else{
        new File("./rEOSBuild/string.s").delete();
    }
}

```

```

        if (vid){
            s+="vid.s\n";
            try{
                Files.copy(
                    new
File("./rEOSSource/vid.s").toPath(),
                    new File("./rEOSBuild/vids").toPath(),
                    StandardCopyOption.REPLACE_EXISTING);
            }
            catch (Exception e){
                return "Build Failed:\nCopying vid.s.";
            }
        }
        else{
            new File("./rEOSBuild/vid.s").delete();
        }
        if (virtMem){
            s+="virtmem.s\n";
            try{
                Files.copy(
                    new
File("./rEOSSource/virtmem.s").toPath(),
                    new
File("./rEOSBuild/virtmem.s").toPath(),
                    StandardCopyOption.REPLACE_EXISTING);
            }
            catch (Exception e){
                return "Build Failed:\nCopying virtmem.s.";
            }
        }
        else{
            new File("./rEOSBuild/virtmem.s").delete();
        }

        return s;
    }

/**
 * Write the standard prologue for all configurations. Use
standard text, except for Display settings and CPU/GPU
 */
private void initText(){

```

```

initText.add("//-----
-----");
    initText.add("// init.s");
    initText.add("//");
    initText.add("// Project:  rEOS 0.1");
    initText.add("//");
    initText.add("// Description: This module is called by
the code in main.s a reflects");
    initText.add("//          the customziations chosen
in the configuration tool.");
    initText.add("//");
    initText.add("// Copyright (c) 2015, 2014 by James J.
Perry");

initText.add("//-----
-----");
    initText.add(".globl initREOS\n");
    initText.add("initREOS:\n");
    initText.add("push {lr}");

    // Set Graphics mode
    if(coreGraphics){
        initText.add("//-----");
        initText.add("// Set-up Display");
        initText.add("//-----");
            // TODO Display height width (default
shown)
            initText.add("mov r0, #"+disp_x);          //
width
            initText.add("mov r1, #"+disp_y);
// height

            // Color Depth
    switch(current.colorMode){
    case CustConfig.CL8Bit:
        initText.add("mov r2, #8");
        break;
    case CustConfig.CL16Bit:
        initText.add("mov r2, #16");
        break;
    case CustConfig.CL24Bit:
        initText.add("mov r2, #24");
        break;

```

```

default:
    initText.add("CM: "+current.displayMode);
}
initText.add("bl vidSetFrameBuffer");

initText.add(" ");
initText.add("// If the display set-up fails,
light the GPIO_OKLED and loop forever.");
initText.add("teq r0,#0 ");
initText.add("bne noError");
initText.add(" ");
initText.add("mov r0,#16 ");
initText.add("mov r1,#1 ");
initText.add("bl setGPIOFunction ");
initText.add("mov r0,#16 ");
initText.add("mov r1,#0 ");
initText.add("bl setGPIO ");
initText.add(" ");
initText.add("error: ");
initText.add("b error ");
initText.add(" ");
initText.add("noError: ");
initText.add("bl setGraphicsADDR ");
initText.add(" ");
initText.add("// Set the default color to green
");
initText.add("mov r0, #0x000007E0 // green
");
initText.add("bl graphicSetFGColor ");
initText.add(" ");
}

// Set Scheduler Mode
initText.add("//-----");
initText.add("// Set-up Scheduler");
initText.add("//-----");
if(current.schedMode==CustConfig.SCH_STATE_spe){
    // SPE
    initText.add("mov r0, #0");
    initText.add("bl schedulerSetMode");
    initText.add(" ");
}

if(current.schedQueueType==CustConfig.SCH_STATE_queue){
    initText.add("mov r0, #0");
}

```

```

        initText.add("bl schedulerSetDS");
    }
    else{
        initText.add("mov r0, #1");
        initText.add("bl schedulerSetDS");
    }
}
else{
    // MPCE
    initText.add("mov r0, #1");
    initText.add("bl schedulerSetMode");
    // Set Quantum
    initText.add("mov r0, #"+quantum);
    initText.add("bl setQuantum");
    // Move mpce.s to build
    mpce=true;
}

// Dump on unsupported
initText.add(" ");
initText.add("//-----");
initText.add("// Dumps/fails ");
initText.add("//-----");

initText.add("ldr r0, =VM_Dump_On_NOP");

if(current.dumpOnUnsupported){
    initText.add("mov r1, #1");
}
else{
    initText.add("mov r1, #0");
}
initText.add("str r1, [r0]");

initText.add("ldr r0, =VM_fail_on_user");

if(current.failOnUserDump){
    initText.add("mov r1, #1");
}
else{
    initText.add("mov r1, #0");
}
initText.add("str r1, [r0]");

```

```

// Dump on unsupported
initText.add(" ");
initText.add("//-----");
initText.add("// Memory Allocation");
initText.add("//-----");

initText.add("ldr r0, =VIRTMEM_first_fit");

if(current.memAllo){
    initText.add("mov r1, #1");
}
else{
    initText.add("mov r1, #0");
}
initText.add("str r1, [r0]");

// Schedule all resident programs
initText.add(" ");
initText.add("//-----");
initText.add("// Schedule rEOS VMR Programs");
initText.add("//-----");

    if (current.schedMode==CustConfig.SCH_STATE_spe){
        for (String s : residentList){
            initText.add("ldr r0, =" + s+"PCB");
            initText.add("bl schedulerSPE ");
        }
    }
    else{
        for (String s : residentList){
            initText.add("ldr r0, =" + s+"PCB");
            initText.add("bl schedulerMPCEAdd ");
        }
    }

// This is the last line of the init.s; it launches
the rESO VMR
initText.add(" ");
initText.add("//-----");
initText.add("// Start rEOS VMR");
initText.add("//-----");

```

```

initText.add("push {r0, r1, r2, lr} ");
initText.add("mov r0, #0x00000041 ");
initText.add("mov r1, #100 ");
initText.add("mov r2, #64");
initText.add("bl drawCharacter");
initText.add("pop {r0, r1, r2, lr} ");

if(current.schedMode==CustConfig.SCH_STATE_spe){
    initText.add("bl schedulerSPERun");
}
else{
//    initText.add("mov r0, #"+quantum);
//    initText.add("bl setQuantum");
    initText.add("mov r0, #"+quantum);
    initText.add("bl schedulerMPCERun");
}
initText.add("pop {lr}");
initText.add("mov pc, lr");

// Write init.s

PrintWriter out=null;
try{
    out = new PrintWriter(new
FileWriter("./reOSSource/init.s"));
}
catch (IOException ioe){

}
for( String s: initText){
    System.out.println(s);
    out.println(s);
}

out.close();

}
}

```

9.2.5 REOSJVMAasm

```

package jjp.reos.reosvm;
import java.io.FileReader;
import java.io.BufferedReader;

```

```

import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
//import java.io.InputStream;
import java.util.ArrayList;

/**
 * This class contains the methods to two-pass
 * assemble an rEOS JVM source file into ARM
 * Assembly .s file that represents the program
 * for compilation in to the build of an
 * rEOS operating system image.
 *
 * @author James J. Perry
 *
 */
public class REOSJVMasm {

    private static int findOpCode(String s){
        for(int x = 0; x<opcodes.length; x++){
            if (opcodes[x].equals(s))return x;
        }
        return -1;
    }

    private static boolean findReserveWord(String s){

        for(int x = 0; x<opcodes.length; x++){
            if (reservedWords[x].equals(s))return true;
        }
        return false;
    }

    private static final boolean[] op2word = {
        false, true, false, false, false,
        false, false, false, false, false,
        false, false, false, false, false,
        false, false, false, false, false,
        false, false, false, false, true,
        true, true, true, true, true,
        true, false, false, false, false,
        false, false, false, false, false
    };
};

```

```

private static final String[] ophex = {
    "00",    "01",    "02",    "03",    "04",
    "05",    "06",    "07",    "08",    "09",
    "0A",    "0B",    "0C",    "0D",    "0E",
    "0F",    "10",    "11",    "12",    "13",
    "14",    "15",    "16",    "17",    "18",
    "19",    "1A",    "1B",    "1C",    "1D",
    "1E",    "1F",    "20",    "21",    "22",
    "23",    "24",    "25",    "26",    "27",
    "28",    "29",    "2A",    "2B",    "2C"
};

private static final String[] opcodes = {
    "", "psh", "dump", "drawCharacter", "drawPixel",
    "halt", "dup", "nop", "pop", "dec", "inc", "add",

    "sub", "time", "and", "or", "swap", "not", "neg",
    "bin2str", "drawString", "mul", "xor", "asl",

    "asr", "div", "drawLine", "fill", "ror", "call",
    "ret", "jmp", "jmpeq", "!@#$", "!@#$$%", "jmpgte",
    "lsr", "mod", "pause", "msgOpen", "msgWrite",
    "msgRead", "gpioRead", "gpioReadSet", "gpioWrite"
};

private static final String[] reservedWords = {
    "_start", "graphicSetFGColor", "setGraphicsADDR",
    "drawPixel", "drawLine", "fillArea", "drawCharacter",
    "getGPIOAddress", "setGPIOFunction", "setGPIO",
    "initREOS", "schedulerMPCEAdd", "schedulerMPCERun",
    "schedulerSetMode", "schedulerSetDS", "schedulerSPE",
    "schedulerSPEAddStack", "schedulerSPEAddQueue",
    "schedulerSPERun", "drawString", "getTimerBase",
    "getTimeStamp", "waitForTimer", "clearMem", "div",
    "mod", "vidMailBoxWrite", "vidMailBoxRead",
    "vidSetFrameBuffer", "VM_main", "font", "ostester",
    "ostesterSTACK", "ostesterPCB", "schedReadyCurr",
    "schedReadyLast", "testcode", "testcodeSTACK",
    "testcodePCB", "testcodea", "testcodeaSTACK",
    "testcodeaPCB", "testcodeb", "testcodebSTACK",
    "testcodebPCB", "testcodec", "testcodecSTACK",
    "testcodecPCB", "frameBufferInfo",
    "VIRTMEM_first_fit", "VM_Dump_On_NOP",
    "VM_fail_on_user"
}

```

```

};

private static BufferedReader in;
private static PrintWriter out;

private ArrayList<String> objArray;
private ArrayList<SymbolTableEntry> symtab;
private ArrayList<String>labellessSource;
private String fn;
private String fpfn;
private int pidNum;
private int pcSpaceSize;
private int sSpaceSize;
private boolean resident;

private static String status = new String();
/**
 * Create an assembler for the rEOS VMR
 */
public REOSJVMassm( ){

}

public String getStatus(){
    return status;
}

/**
 * A private utility used to show listing at the
 * end of assembly.
 *
 * TO DO :: make this utility private after testing.
 *
 * @param s object code array.
 * @param n executable code array.
 */
@SuppressWarnings("unused")
private static void showListing(String[]s, int[]n){
    for (int x = 0; x<s.length; x++){
        System.out.println(n[x]+"\\t"+s[x]);
    }
}

/**
 * A private utility used to save listing at

```

```

    * the end of assembly.
    *
    * TO DO :: Convert to the creation of a file before
release.
    *
    * @param s object code array.
    * @param n executable code array.
    */
@SuppressWarnings("unused")
private static void saveListing(String[]s, int[]n){
    for (int x = 0; x<s.length; x++){
        System.out.println(n[x]+"\\t"+s[x]);
    }
}

/**
 * Reads in a file for assembly.
 *
 * TO DO: Convert to simpler fileio code.
 *
 * @param file path and file to assemble.
 * @return Returns an array of the strings
 * by line) stored in the input file.
 */
private static String[] read (String file){

    ArrayList<String> al = new ArrayList<String>();
    String[] source = new String[1];
    try{
        in = new BufferedReader(new FileReader(file));
    }
    catch (IOException ioe){
        System.out.println("ERROR: reading the saved
file.");
        status = "ERROR: reading the saved file.";
        return source;
    }
    status = "OK";

    String l;
    try{
        while ((l = in.readLine()) != null) {
            al.add(l);
        }
    }

```

```

    }
    catch (IOException ioe){
        System.out.println("ERROR:  reading the saved
file.");
        status = "ERROR:  reading the saved file.";
    }

    try{
        in.close();
    }
    catch (IOException ioe){
        System.out.println("ERROR:  reading the saved
file.");
        status = "ERROR:  reading the saved file.";
    }

    source = al.toArray(source);

    status = "read";
    return source;
}

private void write (){

    try{
        out = new PrintWriter(
            new
FileWriter("./rEOSBuild/"+this.fn+".s"));
    }
    catch (IOException ioe){
        System.out.println("ERROR:  writing the .s
file.");
        status = "ERROR:  writing the .s file.";
        return;
    }
    status = "OK";

    out.println(".data");
    out.println(".globl  "+fn);
    out.println(fn+":");

    for(String s : objArray){
        out.println(s);
    }
}

```

```

    }

    out.close();

    status = "write";

}

/**
 * Checks the validity of tokens and aborts
 * assembly if problem is found.
 *
 * @param s token to check for validity.
 * @param lineNum Line number of assembly
 * line of code being checked.
 * @param line Full line containing token
 * @return true if all tokens are valid
 * and false otherwise
 */

private boolean validTokenCheck(String s, int lineNum,
String line){
    if (findReserveWord(s)){
        System.out.println("REOSCustomizer PASS 1: Line
number "
                        +lineNum+":\n \""+line
                        +"\" contains a reserved word. Assembly
aborted.");
        status = "REOSCustomizer PASS 1: Line number "
                +lineNum+":\n \""+line+
                "\" contains a reserved word. Assembly
aborted.";
        return false;
    }
    if (s.contains(" "))
        || s.contains("(")
        || s.contains(";"){
        System.out.println("REOSCustomizer PASS 1: Line
number "
                        +lineNum+":\n \""+line
                        +"\" contains an illegal space or character.
Assembly aborted.");
        status = "REOSCustomizer PASS 1: Line number "+
                lineNum+":\n \""+line

```

```

        +"\\" contains an illegal space or character.
Assembly aborted.";
        return false;
    }
    return true;
}
/**
 * Symbol Table look-up function
 * @param s Symbol to look up
 * @return numerical value of symbol (if found) or -1 if
not in symbol table.
 */
private int findSymbol(String s, int n){

    for (int x = 0; x<symtab.size(); x++){
        if (symtab.get(x).symbol.equals(s)){
            symtab.get(x).crossref +=n+" ";
            return symtab.get(x).line;
        }
    }
    System.out.println("Symbol not found in symtab");
    status = "Symbol not found in symtab";
    return -1;
}

private String findLabel(int n){

    for (SymbolTableEntry s: symtab){
        if (s.line==n){
            return s.symbol+": ";
        }
    }
    return " ";
}

/**
 * Performs full assemble on a source file.
 *
 * TO DO change to create a file instead of array,
 *
 * @param s Path and file to assemble

```

```

*/
public String assemble(String sp,String sf,
                        String spid, int sSpaceReq,
                        boolean r){
    sSpaceSize = sSpaceReq;

    try{
        pidNum = Integer.parseInt(spid.trim());
    }
    catch(NumberFormatException e){
        status = "Invalid PID number requested";
        System.out.println(status);
        return "";
    }

    if(sf.contains(".")){
        status = "Source files must not have an
extension."
                +" Assembly Aborted";
        System.out.println(status);
        return "";
    }
    fpfn = sp+sf;
    fn = sf;
    String[] s = read(fpfn);

    if(status.equals("ERROR: reading the saved file.)){
        return null;
    }

    return assemble(s,r);
}
/**
 * Method to perform a full assemble of a string of source
code.
 * @param source

*/
public String assemble(String[] source, boolean r){
    resident = r;
    pcSpaceSize=0;
    status = "";
    if(!assml(source)){
        return "Assembly Failed";
    }
}

```

```

    }

    return assm2 ();
}
/**
 * Pass 1 of assemble constructs symbol table and labelless
 * source as input to Pass 2, and finds some syntax
 * and other errors.
 * @param source array
 * @return true on pass 1 success and false otherwise.
 */
public boolean assm1 (String [] source) {
    symtab = new ArrayList <SymbolTableEntry> ();
    labellessSource = new ArrayList <String> ();
    String [] tokens;
    int wordNum=0;
    int lineNum=0;
    for (String s : source) {
        System.out.println ("Line number:" + lineNum + ",
String:" + s);
        s = s.trim ();
        if (s.charAt (0) == '#') {
            labellessSource.add (s);
            continue;
        }

        tokens = s.split (":");
        if (tokens == null) {
            System.out.println ("REOSCustomizer PASS 1: "
                + " Blank lines are not permitted.
Assembly aborted.");
            status = "REOSCustomizer PASS 1: "
                + "Blank lines are not permitted.
Assembly aborted.";
            return false;
        }
        else if (tokens.length == 0) {
            System.out.println ("REOSCustomizer PASS 1:
"
                + "Blank lines are not permitted.
Assembly aborted.");
            status = "REOSCustomizer PASS 1: Blank
lines are not "
                + "permitted. Assembly aborted.";

```

```

        return false;
    }
    else if (tokens.length==1){
        if(!this.validTokenCheck(tokens[0], lineNum,
s)) {
            return false;
        }
        labellessSource.add(tokens[0].trim());
        if (tokens[0].contains(",")){
            wordNum+=2;
        }
        else wordNum++;
        //no pass one work, because labels are not
permitted on blank lines.
    }
    else{
        if(!this.validTokenCheck(tokens[0], lineNum,
s)) {
            return false;
        }
        symtab.add(new
SymbolTableEntry(tokens[0].trim(), wordNum*4));

        labellessSource.add(tokens[1].trim());
        if (tokens[1].contains(",")){
            wordNum+=2;
        }
        else wordNum++;
    }
    lineNumber++;
}
System.out.print("WordNum: "+wordNum);
return true;
}

/**
 * Pass 2 constructs executable array, and finds some
syntax errors.
 * @return An array of integer values that can be executed
by the GPVM
 */
public String assm2(){

```

```

String[] source = new String[labellessSource.size()];
labellessSource.toArray(source);
objArray = new ArrayList<String>();
for(int x = 0; x<source.length; x++){
    source[x] = source[x].trim();
    if(source[x].charAt(0)=='#'){
        objArray.add(source[x]);
        continue;
    }
    String[] tokens =source[x].split(",");
    for (String tk : tokens){
        System.out.println("("+tk+"");
    }
    String[] op=tokens[0].split("#");
    int temp =findOpCode(op[0].trim());
    System.out.println(temp);
    if (temp>=0){
        objArray.add(".int 0x000000"+ophex[temp]+
            "        //
"+this.findLabel(x)+opcodes[temp]);
        this.pcSpaceSize++;
        if(REOSJVMasm.op2word[temp]){
            this.pcSpaceSize++;
        }
        else{
            continue;
        }
    }
    else{
        System.out.println("REOSCustomizer PASS 2:
Line number "
            +x+" contains a invalid opCode.
Assembly Aborted.");
        System.out.println(source[x]);
        status = "REOSCustomizer PASS 2: Line number
"
            +x+" contains a invalid opCode.
Assembly Aborted.";
        return "Assembly Failed";
    }

    if(tokens.length==2){

```

```

String[] tempC = tokens[1].split("#");
String comment="";
if(tempC.length == 2){
    comment = "\t#" + tempC[1];
    tokens[1]= tempC[0];
}

tokens[1]=tokens[1].trim();
boolean isnum=true;
int n =0;
String operand = "";

try{
    n = Integer.parseInt(tokens[1]);
}
catch(NumberFormatException e){
    isnum = false;
}
String hStr = "";

if(isnum){
    hStr = Integer.toHexString(n);
    operand = "    //"
"/**+findLabel(x)*/+hStr;
}
else if
(ConstTable.getConst(tokens[1])!=""){

operand=ConstTable.getConst(tokens[1])+"    //"    ";
    System.out.println(operand);
    hStr="123456789";
}
else {
    hStr =
Integer.toHexString(this.findSymbol(tokens[1], x));
    operand = "    //"
"/**+findLabel(x)*/+tokens[1]+"");
}
operand += comment;

switch (hStr.length()){
case 0:
    objArray.add(".int
0x00000000"+operand);

```

```

        break;
    case 1:
        objArray.add(".int
0x0000000"+hStr+operand);
        break;
    case 2:
        objArray.add(".int
0x000000"+hStr+operand);
        break;
    case 3:
        objArray.add(".int
0x00000"+hStr+operand);
        break;
    case 4:
        objArray.add(".int
0x0000"+hStr+operand);
        break;
    case 5:
        objArray.add(".int
0x000"+hStr+operand);
        break;
    case 6:
        objArray.add(".int
0x00"+hStr+operand);
        break;
    case 7:
        objArray.add(".int
0x0"+hStr+operand);
        break;
    case 8:
        objArray.add(".int
0x"+hStr+operand);
        break;
    case 9:
        objArray.add(".int "+operand);
        break;
    default:
        System.out.println("REOSCustomizer
PASS 2: Line number "+x+" contains a invalid literal number.
Assembly Aborted.");
        status = "REOSCustomizer PASS 2:
Line number "+x+" contains a invalid literal number. Assembly
Aborted.";
        return "Assembly Failed";

```

```

        }

    }

    objArray.add("");
    objArray.add("//---Symbol table and Xref
table-----");
    for (SymbolTableEntry entry : symtab ){
        objArray.add("// "+entry.symbol+":
"+entry.line+"\t\t| "+entry.crossref);
    }
    status = "Assembly complete.";

objArray.add("//-----")
;

    objArray.add("");
    objArray.add(".globl "+fn+"STACK");
    objArray.add(fn+"STACK:");
    objArray.add(".space "+sSpaceSize);

    String pcb = new String("");

    objArray.add(" ");
    objArray.add(".globl " +fn+"PCB");
    objArray.add(fn+"PCB:");

    objArray.add(".int "+pidNum+"                //
PCB_pid ");
    pcb+="PCB_state: \t.int 0x00000000\n";
    objArray.add(".int 0x00000000                //
PCB_state ");
    if(resident){
        pcb+="PCB_pspace: \t.int "+fn+" \n";
        objArray.add(".int "+fn+"
// PCB_pspace ");
    }
    else{
        pcb+="PCB_pspace:\t.int 0x00000000\n";
        objArray.add(".int 0x00000000
// PCB_pspace ");
    }
    pcb+="PCB_psl:\t.int " +(4*this.pcSpaceSize)+"\n";

```

```

        objArray.add(".int " + (4*this.pcSpaceSize)+"
// PCB_psl ");
        if(resident){
            pcb+="PCB_sspace:\t.int "+fn+"STACK\n";
            objArray.add(".int "+fn+"STACK
// PCB_sspace ");
        }
        else{
            pcb+="PCB_sspace:\t.int 0x00000000\n";
            objArray.add(".int 0x00000000
// PCB_sspace ");
        }
        pcb+="PCB_spl:\t.int "+sSpaceSize+"\n";
        objArray.add(".int "+(4*sSpaceSize)+"
// PCB_spl ");
        pcb+="PCB_lpc:\t.int 0x00000000\n";
        objArray.add(".int 0x00000000
// PCB_lpc");
        pcb+="PCB_lsp:\t.int 0x00000000\n";
        objArray.add(".int "+(4*this.sSpaceSize)+"
// PCB_lsp");
        pcb+="PCB_cycles:\t.int 0x00000000\n";
        objArray.add(".int 0x00000000
// PCB_cycles");
        pcb+="PCB_source:\t.int 0x00000000\n";
        objArray.add(".int 0x00000000
// PCB_source");
        pcb+="PCB_target:\t.int 0x00000000\n";
        objArray.add(".int 0x00000000
// PCB_target");
        pcb+="PCB_next:\t.int 0x00000000\n";
        objArray.add(".int 0x00000000
// PCB_next");
        pcb+="PCB_prev:\t.int 0x00000000\n";
        objArray.add(".int 0x00000000
// PCB_prev");

        System.out.println("object code:");
        write();
        for(String s : objArray){
            System.out.println (s);
        }
        status="Success";

```

```

        return pcb;
    }
}

```

9.2.4 SymbolTableEntry

```

package jjp.reos.reosvm;

public class SymbolTableEntry {
    String symbol;
    int line;
    String crossref = "";
    public SymbolTableEntry(String s, int n){
        symbol = s;
        line = n;
    }
}

```

9.3 Samples of customizer generated code9.3.1 config.txt

```

#####
#####
# Memory Settings
#
# Note: rEOS configuration tools do not support CMA Dynamic
Memory Split
#     because it is not officially supported.
#####
#####
gpu_mem=128
disable_l2cache=0
disable_pvt=0
#####
#####
# Display Settings
#####
#####
#####
#####
# Boot Settings
#####
#####

```

9.3.2 init.s (SPE)

The init.s example below was generated as part of the SPE scheduling mode build.

```
//-----
--
// init.s
//
// Project:  rEOS 0.1
//
// Description: This module is called by the code in main.s a
reflects
//           the customziations chosen in the configuration
tool.
//
// Copyright (c) 2015, 2014 by James J. Perry
//-----
--
.globl initREOS

initREOS:

push {lr}
//-----
// Set-up Scheduler
//-----
mov r0, #0
bl schedulerSetMode

mov r0, #0
bl schedulerSetDS

//-----
// Dumps/fails
//-----
ldr r0, =VM_Dump_On_NOP
mov r1, #1
str r1, [r0]
ldr r0, =VM_fail_on_user
mov r1, #1
str r1, [r0]
```

```

//-----
// Memory Allocation
//-----
ldr r0, =VIRTMEM_first_fit
mov r1, #1
str r1, [r0]

//-----
// Schedule rEOS VMR Programs
//-----
ldr r0, =ostesterPCB
bl schedulerSPE

//-----
// Start rEOS VMR
//-----
push {r0, r1, r2, lr}
mov r0, #0x00000041
mov r1, #100
mov r2, #64
bl drawCharacter
pop {r0, r1, r2, lr}
bl schedulerSPERun

```

9.4 rEOS VMR Program Examples

The programs in this section demonstrate examples and tests of rEOS VMR assembly code.

9.4.1 ostester

This program tests the various rEOSVMR wordcodes.

```

#-----
# Test psh, drawCharacter
# --Expect "A" at (0,20) in Red
#-----
        psh, 65          # A
        psh, 0           # y Pos

```

```

    psh, 20          # x Pos
    psh, Color.red  # Draw in red.
    drawCharacter  # Draw the A to the screen
#-----
# Test psh, drawLine
# --Expect line from (20,50, 20, 80) in Red
#-----
    psh, 50          # y1 Pos
    psh, 20          # x1 Pos
    psh, 80          # y0 Pos
    psh, 20          # x0 Pos
    psh, Color.red  # Draw in red.
    drawLine       # Draw the A to the screen
#-----
# Test psh, fill
# --Expect Fill rectangle at (10, 40, 20, 20)
#-----
    psh, 20          # y1 Pos
    psh, 20          # x1 Pos
    psh, 40          # y0 Pos
    psh, 10          # x0 Pos
    psh, Color.red  # Draw in red.
    fill           # Draw the A to the screen
#-----
# Test psh, drawString
# -- Expect "HELLO" at (20, 20) in Red
#-----
    psh, 79          # O
    psh, 76          # L
    psh, 76          # L
    psh, 69          # E
    psh, 72          # H
    psh, 5           # string length
    psh, 20          # y Pos
    psh, 20          # x Pos
    psh, Color.red  # Draw in red.
    drawString      # Draw
#-----
# Test psh, bin2str, drawString conversion
# -- Expect "4" at (20, 60) in Red
#-----
#           psh, 4          # Push the data
#           bin2str        # Convert to ascii
#           psh, 60        # y Pos

```

```

#      psh, 20          # x Pos
#      psh, Color.red  # Draw in red.
#      drawString      # Draw
#-----
# Test the add, psh, bin2str, drawString
# -- Expect "8" at (20, 80) in red
#-----
#      psh, 3          # Push 3 data
#      psh, 5          # Push 5 data
#      add             # Add operands
#      bin2str        # convert to Ascii
#      psh, 80         # y Pos
#      psh, 20         # x Pos
#      psh, Color.red  # Draw in red.
#      drawString     # Draw
#-----
# Test the and, psh, bin2str, drawString
# -- Expect 1 at (20, 100) in red
#-----
#      psh, 3          # Push 3 data
#      psh, 5          # Push 5 data
#      and            # and operator
#      bin2str        # convert to Ascii
#      psh, 100        # y Pos
#      psh, 20         # x Pos
#      psh, Color.red  # Draw in red.
#      drawString     # Draw
#-----
# Test the asl, psh, bin2str, drawString
# -- Expect 12 at (20, 120) in blue
#-----
#      psh, 3          # Push 3 data
#      psh, 2          # Push 5 data
#      asl            # Add operands
#      bin2str        # convert to Ascii
#      psh, 120       # y Pos
#      psh, 20         # x Pos
#      psh, Color.blue # Draw in blue.
#      drawString     # Draw
#-----
# Test the asr, psh, bin2str, drawString
# -- Expect 1 at (20, 140) in blue
#-----
#      psh, 3          # Push 3 data

```

```

    psh, 2          # Push 5 data
    asr            # Asr operands
    bin2str        # convert to Ascii
    psh, 140      # y Pos
    psh, 20       # x Pos
    psh, Color.blue # Draw in blue.
    drawString    # Draw
#-----
# Test the dec, psh, bin2str, drawString
# -- Expect 2 at (20, 160) in blue
#-----
    psh, 3          # Push 3 data
    dec            # decrement
    bin2str        # convert to Ascii
    psh, 160      # y Pos
    psh, 20       # x Pos
    psh, Color.blue # Draw in blue.
    drawString    # Draw
#-----
# Test the dup, psh, bin2str, drawString
# -- Expect 3 at (20, 180) in blue
# -- Expect 3 at (30, 180) in green
#-----
    psh, 3          # Push 3 data
    dup            # duplicate
    bin2str        # convert to Ascii
    psh, 180      # y Pos
    psh, 20       # x Pos
    psh, Color.blue # Draw in blue.
    drawString    # Draw
    bin2str        # convert to Ascii
    psh, 180      # y Pos
    psh, 30       # x Pos
    psh, Color.green # Draw in green.
    drawString    # Draw
#-----
# Test the inc, psh, bin2str, drawString
# -- Expect 4 at (20, 200) in blue
#-----
    psh, 3          # Push 3 data
    inc            # increment
    bin2str        # convert to Ascii
    psh, 200      # y Pos
    psh, 20       # x Pos

```

```

        psh, Color.blue # Draw in blue.
        drawString # Draw
#-----
# Test the mul, psh, bin2str, drawString
# -- Expect 6 at (20, 220) in blue
#-----
        psh, 3          # Push 3 data
        psh, 2          # Push 5 data
        mul             # mul operand
        bin2str         # convert to Ascii
        psh, 220       # y Pos
        psh, 20        # x Pos
        psh, Color.blue # Draw in blue.
        drawString # Draw
#-----
# Test the sub, psh, bin2str, drawString
# -- Expect -1 at (20, 240) in blue
#-----
        psh, 4          # Push 3 data
        psh, 5          # Push 5 data
        sub             # sub 4-5 operand
        bin2str         # convert to Ascii
        psh, 240       # y Pos
        psh, 20        # x Pos
        psh, Color.blue # Draw in blue.
        drawString # Draw
#-----
# Test the neg, psh, bin2str, drawString
# -- Expect -3 at (20, 260) in blue
#-----
        psh, 3          # Push 3 data
        neg             # negate
        bin2str         # convert to Ascii
        psh, 260       # y Pos
        psh, 20        # x Pos
        psh, Color.blue # Draw in blue.
        drawString # Draw
#-----
# Test the not, psh, bin2str, drawString
# -- Expect -6 at (20, 280) in blue
#-----
        psh, 5          # Push 5 data
        not             # Bitwise not
        bin2str         # convert to Ascii

```

```

    psh, 280 # y Pos
    psh, 20 # x Pos
    psh, Color.blue # Draw in blue.
    drawString # Draw
#-----
# Test the or, psh, bin2str, drawString
# -- Expect 3 at (300, 0) in blue
#-----
    psh, 3 # Push 3 data
    psh, 2 # Push 2 data
    or # or operand
    bin2str # convert to Ascii
    psh, 0 # y Pos
    psh, 300 # x Pos
    psh, Color.blue # Draw in blue.
    drawString # Draw
#-----
# Test the pop, psh, bin2str, drawString
# -- Expect 2 at (300, 20) in blue
#-----
    psh, 3 # Push 3 data
    psh, 2 # Push 2 data
    pop # pop off the 2
    bin2str # convert to Ascii
    psh, 20 # y Pos
    psh, 300 # x Pos
    psh, Color.blue # Draw in blue.
    drawString # Draw
#-----
# Test the ror, psh, bin2str, drawString
# -- Expect 1 at (300, 40) in blue
#-----
    psh, 2 # Push 2 data
    psh, 1 # Push 1 data
    ror # or operand
    bin2str # convert to Ascii
    psh, 40 # y Pos
    psh, 300 # x Pos
    psh, Color.blue # Draw in blue.
    drawString # Draw
#-----
# Test the xor, psh, bin2str, drawString
# -- Expect 1 at (300, 60) in blue
#-----

```

```

    psh, 3          # Push 3 data
    psh, 2          # Push 2 data
    xor            # or operand
    bin2str        # convert to Ascii
    psh, 60         # y Pos
    psh, 300       # x Pos
    psh, Color.blue # Draw in blue.
    drawString     # Draw
#-----
# Test the swap, psh, bin2str, drawString
# -- Expect 2 3 at (300, 80) in blue
#-----
    psh, 3          # Push 3 data
    psh, 2          # Push 2 data
    swap          # or operand
    bin2str        # convert to Ascii
    psh, 80         # y Pos
    psh, 300       # x Pos
    psh, Color.blue # Draw in blue.
    drawString     # Draw
    bin2str        # convert to Ascii
    psh, 80         # y Pos
    psh, 310       # x Pos
    psh, Color.blue # Draw in blue.
    drawString     # Draw
#-----
# Test the sub, psh, bin2str, drawString
# -- Expect -1 at (300, 100) in blue
#-----
    psh, 4          # Push 4 data
    psh, 5          # Push 5 data
    sub           # sub 4-5 operand
    bin2str        # convert to Ascii
    psh, 100       # y Pos
    psh, 300       # x Pos
    psh, Color.blue # Draw in blue.
    drawString     # Draw
#-----
# Test the div, psh, bin2str, drawString
# -- Expect 0 at (300, 120) in blue
#-----
    psh, 4          # Push 4 data
    psh, 5          # Push 5 data
    div           # div 4/5 operand

```

```

    bin2str      # convert to Ascii
    psh, 120    # y Pos
    psh, 300    # x Pos
    psh, Color.blue # Draw in blue.
    drawString  # Draw
#-----
# Test the mod, psh, bin2str, drawString
# -- Expect 4 at (300, 140) in blue
#-----
    psh, 4      # Push 4 data
    psh, 5      # Push 5 data
    mod         # mod 4/5 operand
    bin2str     # convert to Ascii
    psh, 140   # y Pos
    psh, 300   # x Pos
    psh, Color.blue # Draw in blue.
    drawString # Draw
#-----
# Test the lsr, psh, bin2str, drawString
# -- Expect 2 at (300, 160) in blue
#-----
    psh, 4      # Push 4 data
    psh, 1      # Push 1 data
    lsr        # lsr operand
    bin2str     # convert to Ascii
    psh, 160   # y Pos
    psh, 300   # x Pos
    psh, Color.blue # Draw in blue.
    drawString # Draw
#-----
# Test the time, bin2str, drawString
# -- Expect ? at (300, 180) in blue
#-----
    time       # getTime
    bin2str    # convert to Ascii
    psh, 180   # y Pos
    psh, 300   # x Pos
    psh, Color.blue # Draw in blue.
    drawString # Draw
#-----
# Test JMP
# -- Fail on J at 300, 200) in red
# -- pass Y at 320, 200
#-----

```

```

jmpst:  jmp, skipj      # Unconditional jump to skipj
        psh, 74         # J
        psh, 200      # y Pos
        psh, 300      # x Pos
        psh, Color.red # Draw in red.
        drawCharacter # Draw the J to the screen
skipj:  psh, 89         # Y
        psh, 200      # y Pos
        psh, 320      # x Pos
        psh, Color.red # Draw in red.
        drawCharacter # Draw the Y to the screen
#-----
# Test JMPEqt
# -- Fail on K at (300, 220) in red
# -- pas Y at 320, 220
#-----
        psh, 1        # push 1
        psh, 1        # push 1
        jmpeq, skipjeqt # Jump if equal.
        psh, 75         # K
        psh, 220      # y Pos
        psh, 110      # x Pos
        psh, Color.red # Draw in red.
        drawCharacter # Draw the K to the screen
skipjeqt: psh, 89      # Y
        psh, 220      # y Pos
        psh, 320      # x Pos
        psh, Color.red # Draw in red.
        drawCharacter # Draw the Y to the screen
#-----
# Test JMPEqf
# -- pass K at (300, 240) in red
# -- pass Y at 320, 240
#-----
        psh, 1        # push 1
        psh, 0        # push 0
        jmpeq, skipjeqf # Jump if equal.
        psh, 75         # K
        psh, 240      # y Pos
        psh, 300      # x Pos
        psh, Color.red # Draw in red.
        drawCharacter # Draw the K to the screen
skipjeqf: psh, 89      # Y
        psh, 240      # y Pos

```

```

        psh, 320 # x Pos
        psh, Color.red # Draw in red.
        drawCharacter # Draw the Y to the screen
#-----
# Test JMPgtet
# -- Fail on L at (300, 260) in red
#-----
        psh, 1 # push 1
        psh, 1 # push 1
        jmpgte, skipjgtet # Jump if equal or greater.
        psh, 76 # L
        psh, 260 # y Pos
        psh, 300 # x Pos
        psh, Color.red # Draw in red.
        drawCharacter # Draw the L to the screen
skipjgtet: psh, 89 # Y
        psh, 260 # y Pos
        psh, 320 # x Pos
        psh, Color.red # Draw in red.
        drawCharacter # Draw the Y to the screen
#-----
# Test JMPgtef
# -- fail on L at (300, 280)
# -- Expect Y at (320, 280) in red
#-----
        psh, 1 # push 1
        psh, 0 # push 0
        jmpgte, skipjgtef # Jump if equal or greater.
        psh, 76 # L
        psh, 280 # y Pos
        psh, 300 # x Pos
        psh, Color.red # Draw in red.
        drawCharacter # Draw the L to the screen
skipjgtef: psh, 89 # Y
        psh, 280 # y Pos
        psh, 320 # x Pos
        psh, Color.red # Draw in red.
        drawCharacter # Draw the Y to the screen
#-----
# Test JMPgtef
# -- Expect L at (300, 300)
# -- Expect Y at (320, 300) in red
#-----
        psh, 0 # push 0

```

```

    psh, 1      # push 1
    jmpgte, skipjgtef # Jump if equal or greater.
    psh, 76     # L
    psh, 300   # y Pos
    psh, 300   # x Pos
    psh, Color.red # Draw in red.
    drawCharacter # Draw the L to the screen
skipjgtef:  psh, 89     # Y
    psh, 300   # y Pos
    psh, 320   # x Pos
    psh, Color.red # Draw in red.
    drawCharacter # Draw the Y to the screen

#-----
# Test NOP
#-----
        nop          # No Operation
#-----
# Test Call
#-----
        call, subr   # call subr
#-----
# Test Halt
#-----
        halt        #
#-----
# Test ret, drawPixel
#-----
subr:   psh, 300     # y Pos
        psh, 200   # x Pos
        psh, Color.red # Draw in red.
        drawPixel  # Draw
        ret          # return to caller

```

10.0 Bibliography

Anonymous. *BOARDCOM. BCM2835 ARM Peripherals*. Broadcom Europe Ltd. 2012

Anonymous. *ABI for the ARM 32-bit Architecture*

<http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.subset.swdev.abi/index.htm>

1 Web 6/14/201.

Anonymous. “Raspberry Pi Schematics”

<https://www.raspberrypi.org/documentation/hardware/raspberrypi/schematics/Raspberry-Pi-Rev-2.1-Model-AB-Schematics.pdf>

Anonymous. “What are the system requirements for Java?”

<http://java.com/en/download/help/sysreq.xml> Web 6/5/2015.

Borman, C. et. al. RFC 7228: *Terminology for Constrained Node Networks*

<https://tools.ietf.org/html/rfc7228>. 2014.

Chadwick, Alex. “Pi: Operating Systems Development” ED Robert Mullins.

<http://www.cl.cam.ac.uk/projects/raspberrypi/tutorials/os/index.html>. 2012.

Heath, Steve. *Embedded Systems Design*. EDN series for design engineers (2 ed.). Newnes.

2013.

Koreschoff, Treffyn Lynch, Toni Roberston, Tuck Wah Leong. “Internet of Things: a Review of

Literature and Products.” OxCHI’13, November 25-29 2013, Adelaide, Australia.

Kovatsch, Matthias. “CoAP for the World of Things: From Tiny Resource-constrained Devices

to the Web of Things.” Proceedings of UbiComp 2013, Zurich, Switzerland.

Perry, James J. "Boot Process Overview."

<https://docs.google.com/presentation/d/1kbfHZbgjvsU5UGkFT6Fk5CmftUWKD4BQJk87Li-l1vY/edit?usp=sharing> 2014.

Perry, James J. "General Purpose Virtual Machine for Beginners."

<http://livingcompsci.com/technical-tutorials-and-code-examples/virtual-machine-tutorials/general-purpose-virtual-machine-for-beginners/>. 2014.

Singhal, Mukesh and Niranjana G. Shivaratri. *Advanced Concepts in Operating Systems:*

Distributed, Database, and Multiprocessor Operating Systems. New York: McGraw-Hill, 1994.

Silberschatz, Abraham, et.al. *Operating System Concepts*. Hoboken, NJ: Wiley, 2014.