Universidade do Minho
Escola de Engenharia

ESRG
EMBEDDED SYSTEMS
RESEARCH
GROUP

Master degree in Industrial Electronic Engineering and Computers
Specialization in Embedded Systems and Computers

VeSPA ABI Specification

Class Project

Professor:
Adriano Tavares

June 2024

Agenda

1 Introduction

1.1 The Importance of an ABI

An Application Binary Interface (ABI) defines a standard set of rules and conventions that govern how program modules interact at the binary level. Among many things, it specifies how functions are called, how data is represented in memory and how exceptions are handled.

Essentially, an ABI serves as a contract between different software and hardware modules, ensuring they can work together, even if they were developed independently. This is of extreme importance because it simplifies the process of porting software to VeSPA-based platforms, ensuring compatibility and reducing future development time.

1.2 Memory Model

The VeSPA SoC implementation follows an Harvard style memory architecture, which defines two different memory regions: One responsible for storing the program binary code, and the other responsible for storing data.

The stack is mapped at the top of the data memory region and it grows downwards as it will defined.

2 | VeSPA Calling Conventions

2.1 Register Calling Conventions

Register	Usage	Caller Saved
R0	Zero Register	-
R1	Return Address	Yes
R2	Frame Pointer	Yes
R3	Stack Pointer	Yes
R4 - R5	Return Registers and special registers for Multiplication / Division	No
R6 - R31	Temporary Registers	No

Table 2.1: Register Usage

The VeSPA architecture provides 32 general purpose registers labeled R0 to R31. As shown in table 2.1, register R0 is hardwired to zero, register R1 contains the return address of functions, register R2 and R3 contain the Frame Pointer and Stack Pointer to manage functions memory. Registers R4 and R5 are used to contain the return values from functions, but are also extended to serve as parameter passing registers to special functions that emulate multiplication and division operations as the baseline ISA does not support those. The remaining 26 registers are temporary registers used for general data manipulation. All registers are global and visible to any procedure active.

2.2 Data Representation

Considering the constraints implied by the underlying hardware, the ABI defines the usage of big-endian byte order, meaning the least significant byte occupies the highest byte address in memory as shown.

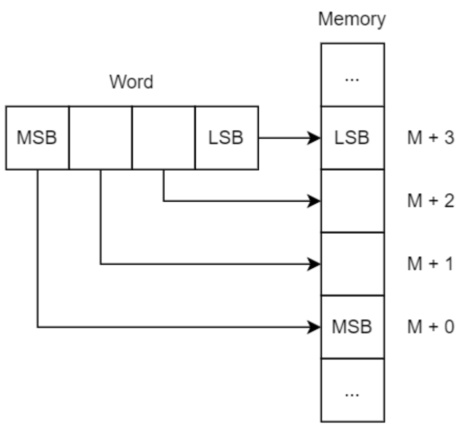

Figure 2.1: Instruction Format

2.2.1 Fundamental Data types

Given the VeSPA arquitecture all C/C++ data types have the same size and alignment of 1 memory space of 4 bytes.

2.2.2 Composite Data types

A composite data type is a collection of one or more primitive data types that are handled as single entity. For now only arrays are supported by the ABI and the following conventions are employed:

- Array
  - Alignment of an array is the same of its base type;
  - Size is the number of elements times the size of its base type.

2.3 Procedure Calling Convention

This section describes the standard calling sequence, including parameter passing, register usage and stack management. It is recommended all functions follow the following sequence when possible.

- Push arguments to the stack;
- Save Frame Pointer;
- Copy current Stack Pointer to Frame Pointer;
- Save Return Address
- Jump to function using the JMPL instruction with R1;
- Acquire local space on the stack;
- Execute function;
- Place result in return value registers;
- Copy current Frame Pointer to Stack Pointer to free stack space;
- Restore Return Address;
- Restore Frame Pointer;
- Restore the program to where it was called, using the JMP instruction;
- Free stack space occupied by function's arguments.

Across functions calls it is not guaranteed that the values stored in registers are preserved with the exception of registers R1, R2 and R3, which are saved by the caller function onto the stack.

Parameter Passing

Parameter passing is done primarily through the stack and the following conventions are adopted:

- All arguments are pushed to the stack with the first argument starting at the first available stack space;
- Return values are passed through register R4;
- There is an exception where the functions that emulate multiplication and division receive their parameters through registers R4 and R5.

Stack Management

- The stack grows downwards towards lower addresses;
- The first argument passed onto the stack is located at offset given by the number of parameters plus two of the stack pointer on function entry;

Stack Frame

Each function has a stack frame that grows downwards from high addresses. The table 2.2 shows the stack organization. The updated frame pointer points to the previous frame pointer, with arguments accessible through positive offsets and the return address and local variable with negative offsets.

Position	Contents	Frame
	...	Previous
R2 + n	Argument 0	Current
	...	
R2 + 1	Argument n	
R2	Previous frame pointer	
R2 - 1	Return Address	
R2 - 2	Local Variable 0	
	...	
R2 - m	Local Variable m	

Table 2.2: Stack Frame Structure

Bitfields

Bitfields are packed in big-endian order.

2.4 Interrupts

Due to the non-deterministic nature of interrupts calls, the procedure calling convention is expanded to preserve the machine status. An interrupt callback function must, in addition to what was stated in section 2.3:

- Preserve all non callee saved registers;

- Use the RETI instruction to restore the execution flow;

Furthermore, to define a interrupt handling function the source code must declare a void type function with no arguments. The compiler will detect these functions signatures and create the necessary instructions at interrupt vectors to jump to these functions. In table 2.3, it is possible to find the supported function names, and interrupts associated with them.

Interrupt	Function
UART Peripheral	void UART_ISR(void)
PS2 Peripheral	void PS2_ISR(void)
Timer Peripheral	void TIMER_ISR(void)
GPIO Peripheral	void GPIO_ISR (void)

Table 2.3: Interrupt Handling Functions