MIS for Knaves NES

Niko Savas	Joe Crozier	Sam Nalwa
1300247	1311502	1332792

November 6, 2015

Software Engineering 3XA3, Lab #3

Contents

1 Introduction

The following document details the module interface specifications for the implemented modules in KnavesNES. It is intended to ease navigation through the program for design and maintenance purposes. This document is meant to be used in tandem with the System Requirements Specifications and the Module Guide.

2 Module Heirarchy

Because the Knaves NES emulator is simply emulating the CPU of an NES, it doesn't require significant module depth. In the larger scope of an NES emulator, the modules included in the CPU emulator would be lower level modules to the general emulator. For this reason, Knaves NES heirarchy is relatively flat, including only one Level 1 module.

Level 1	Level 2
Command Line Interfacing Module	CPU Module
-	Cartridge Module
-	Memory Module

Table 1: Module Heirarchy for Knaves NES

3 MIS of Command line Interface Module

3.1 Exported Access Programs

Name	In	Out	Exceptions
init	string - filename	-	File Not Found
run	-	-	-
stop	-	-	-

Table 2: Exported Access Programs for Command Line Interface Module

3.2 Interface Semantics

3.2.1 State Variables

isRunning: BOOL

cycles: int

3.2.2 Environment Variables

_cpu: CPU

_memory: Memory

3.2.3 Assumption

The method calls will be made in the following order: init, run stop.

3.2.4 Access Program Semantics

Init will initialize the passed cartridge file. Run will start emulation of the CPU, stop will stop the current emulation.

4 MIS of Memory Module

4.1 Exported Access Programs

Name	In	Out	Exceptions
write	unsigned short address, unsigned char value	-	Invalid address, invalid char
read	unsigned short address	unsigned char value	Invalid address
logMemory	-	-	Unable to write file

Table 3: Exported Access Programs for Memory Module

4.2 Interface Semantics

4.2.1 State Variables

RAM: char[]

4.2.2 Environment Variables

4.2.3 Assumption

Reads and writes will be made to addresses within the scope of the memory, otherwise an exception will be called.

4.2.4 Access Program Semantics

5 MIS of Cartridge Module

5.1 Exported Access Programs

Name	In	Out	Exceptions
main	string: FileName	InstructionSet	File Not Found

Table 4: Exported Access Programs for Cartridge Module

5.2 Interface Semantics

5.2.1 State Variables

5.2.2 Environment Variables

FileName: Name of the ROM file to be read

5.2.3 Assumption

5.2.4 Access Program Semantics

The main function will take the filename given as an input, and read through the ROM file and extract the necessary information and instruction set from it.

6 MIS of CPU Module

6.1 Exported Access Programs

Name	In	Out	Exceptions
init	-	-	File Not Found
start	-	-	-
reset	-	-	-
executeInstruction	-	unsigned short - cycles	Invalid Instruction
executeInterrupt	enum - interrupt	-	Invalid Interrupt
getSource	Mode- mode	unsigned short - operand	-
checkInterrupts	-	BOOL - interruptRun	-
readAddress	unsigned short - address	unsigned char - value	Invalid Address
pushStack	unsigned char byte	-	-
hasStatus*	unsigned char flag	BOOL - reg_status	-
updateStatus*	unsigned short val	-	-
funcLoadAccumulator	-	-	-
funcAddWithCarry	-	-	-
funcBranchNotEqualZero	-	-	-
funcCompareMemory	-	-	-
funcStoreAccumulator	-	-	-
funcTransferAccumulatorToX	-	-	-

Table 5: Exported Access Programs for CPU Module

6.2 Interface Semantics

6.2.1 State Variables

isRunning: BOOL

reg_pc: int

reg_status: int

reg_x: int

reg_y: int

instructions: Enum

6.2.2 Environment Variables

_cpu: CPU

_memory: Memory

6.2.3 Assumption

6.2.4 Access Program Semantics

Init will initialize the CPU. Start will begin CPU emulation. Reset will reset the CPU to its original state. ExecuteInstruction will execute the instruction referenced at the current reg_pc position memory. ExecuteInterrupt will execute the passed interrupt. GetSource will get the source of the operand from the passed mode. CheckInterrupts will check if an interrupt is required to run. ReadAddress will use the Memory module to read the memory value from the passed address. PushStack will push the passed char to the stack. hasStatus* methods will check if the CPU status register includes a given status. ClearStatus* methods will clear the passed flag from

the CPU status. UpdateStatus* will update the respective status register. FuncLoadAccumulator will load a value from the referenced address to the accumulator.

Revision History

Revision	Date	Author(s)	Description
1.0	Nov 5, 2015	Niko Savas	created