**Writing a complete Snake game**

**in AQA Assembly Language**

TEACHER’S GUIDE

Created by Richard Pawson

[Creative Commons License](https://creativecommons.org/licenses/by-sa/4.0/)  
This work is licensed under a [Creative Commons Attribution-ShareAlike 4.0 International License](https://creativecommons.org/licenses/by-sa/4.0/).

[Introduction 3](#_Toc504677135)

[Background 3](#_Toc504677136)

[The implementation 3](#_Toc504677137)

[Important note on addressing modes 3](#_Toc504677138)

[How to use this resource 3](#_Toc504677139)

[Complete code after Story 1 5](#_Toc504677140)

[Complete code after Story 2 6](#_Toc504677141)

[Complete code after Story 3 7](#_Toc504677142)

[Complete code after Story 4 8](#_Toc504677143)

[Complete code after Story 5 9](#_Toc504677144)

[Complete code after Story 6 10](#_Toc504677145)

[Complete code after Story 7 11](#_Toc504677146)

[Complete code after Story 8 12](#_Toc504677147)

[Complete code after Story 9 13](#_Toc504677148)

[Complete code after Story 10 14](#_Toc504677149)

[Complete code after Story 11 15](#_Toc504677150)

[Complete code after Story 12 17](#_Toc504677151)

[Complete code after Story 13 19](#_Toc504677152)

[Complete code after Story 14 21](#_Toc504677153)

[Complete code after Story 15 23](#_Toc504677154)

[Complete code after Story 16 25](#_Toc504677155)

[Complete code after Story 17 27](#_Toc504677156)

# Introduction

This Teacher Guide is to be used in conjunction with the Student Workbook of the same title.

## Background

This resource was created as a response to the difficulty of engaging pupils with assembly language programming when using only trivial examples such as adding two numbers, or finding the largest of three values. I wanted, instead, to give pupils the experience of writing a program that:

* Was non-trivial in difficulty (of the order of 100 machine instructions)
* Delivered a rewarding result
* Was appropriate to be written in assembly language

Most people who’ve played a few video games will have encountered ‘Snake’ at some point - probably on a PC or phone. Despite its simplicity it is a very engaging game, and many pupils will be keen to play it, and compete to achieve the longest snake.

Today’s pupils are just about old enough to remember simple hand-held video games that featured low-resolution liquid crystal displays. Those devices typically had 8-bit microprocessors and *very limited* memory. The game on those devices, like Snake - all simple by today’s standards - had to be written in assembly language both for speed of execution and size.

## The implementation

The implementation presented uses less than 100 machine-level instructions. It is written for the AQA Assembly Language, which is designed for an imaginary Risc processor loosely based on an ARM processor.

The Snake program will run on the AQA Processor Simulation written by Peter Higginson (best known in school circles for his online LMC simulator), and available online here: <http://www.peterhigginson.co.uk/AQA/> . Peter extended his simulation specifically to make it possible to run this Snake game, and provided lots of helpful advice in the coding of the game itself.

It would not take a great deal of effort to modify the final code to run on a real ARM processor, as used on the Rasberry Pi, equipped with a suitable memory-addressable display.

## What this resource helps to teach

* Assembly Language programming (using AQA specification)
* Immediate, Direct, Indexed, and Indirect Addressing Modes (see note below)
* Agile software development methodology
* Linear and Circular Queue data structures
* Simple game design

## Important note on addressing modes

The AQA specification for assembly language covers only the *immediate* and *direct* addressing modes: it does not cover *indexed*, or *indirect* addressing modes used in this Snake game – even though the ARM processor (from which the AQA specification has clearly been derived) does have those modes, as do most processors, old and new. In fact, you can’t write *any* interesting programs in assembly language without either indexed or indirect modes, unless, that is, you resort to writing self-modifying programs, which is not considered good practice and which many modern computers prevent for security reasons. Peter’s simulator does include indexed and indirect addressing modes, using a syntax he has designed to be compatible/consistent with the standard AQA specification.

We believe it better that pupils learn all four of the *main* addressing modes (some processors offer even more) – immediate, direct, indexed, and indirect. But they need to remember, in answering an AQA exam question, that the AQA spec covers only the first two.

## How to use this resource

The Student Workbook consists of 17 ‘stories’ – each of which involves writing no more than 10 machine operations, and, importantly, each of which (well, almost each) delivers incremental value that can be tested. It thus follows the modern, ‘agile’, approach to software development.

Each story starts with a short statement of the requirement for that story, followed, where appropriate, by a brief discussion of algorithms and/or techniques that are appropriate to its implementation. It then presents the new lines of code to be written and/or existing lines that need to be changed. Finally, many stories include some questions that the pupil should answer before proceeding to the next story.

The Student Workbook and this Teacher’s Guide are both Word documents, so you may choose to edit or extend the stories. For more able pupils, you might even prefer not to provide all of the necessary code changes in each case, and instead require the pupils to come up with the new/changed lines of code, or some of them, for themselves - at least for the simpler stories.

In this Teacher Guide you will find a complete version of the code as it should be at the end of each story in the worksheet. Thus the code for Story 17 in the Teacher’s Guide is the finished solution, at least as far as the worksheet is concerned: some students may well want to go on to refining or extending the game with their own stories.

You might like to start by demonstrating – from the front - the finished game, using the Story 17 finished code. This will give the students a sense of what they are aiming for - many will find this quite motivating. But it is recommended that you do *not* give out the complete code to the students up front.

The intent in providing the complete code at the end of each story is to allow you to:

* Help a student that has got completely stuck, or made a mess of their code without having a saved version from their previous story.
* Skip through some of the stories if there isn’t sufficient time available to complete the whole sheet

The Student Workbook is intended to be sufficiently comprehensive that students could complete the exercise without intervention and/or in their own time. But getting all the students through the first few stories in a classroom can be helpful.

# Complete code after Story 1

defineRegisters:

mov r1,#0x008844 //Snake colour (green)

drawSnake:

str r1,527 //Tail

str r1,528 //Head

# Complete code after Story 2

defineRegisters:

mov r1,#0x008844 //Snake colour

mov r2,#0xffffff //Background colour (white)

mov r3, #527 //Tail position, initialised

mov r4, #528 //Head position, initialised

drawSnake:

str r1,[r3] //Tail

str r1,[r4] //Head

moveSnake:

str r2,[r3] //Reset tail to Background

add r3,r3,#1 //Increment the tail pointer by 1

add r4,r4,#1 //Increment the head pointer by 1

str r1,[r4] //Draw new head

b moveSnake //Loop

# Complete code after Story 3

defineRegisters:

mov r1,#0x008844 //Snake colour (green)

mov r2,#0xffffff //Background colour (white)

mov r3, #271 //Tail position, initialised

mov r4, #272 //Head position, initialised

drawSnake:

str r1,[r3+256] //Tail

str r1,[r4+256] //Head

moveSnake:

str r2,[r3+256] //Reset tail to Background

add r3,r3,#1 //Increment the tail pointer by 1

add r4,r4,#1 //Increment the head pointer by 1

str r1,[r4+256] //Draw new head

b moveSnake //Loop

# Complete code after Story 4

defineRegisters:

mov r1,#0x008844 //Snake colour (green)

mov r2,#0xffffff //Background colour (white)

mov r3, #271 //Tail position, initialised

mov r4, #272 //Head position, initialised

mov r5, #520 //Apple position

mov r6, #0xff8800 //Apple colour

drawSnake:

str r1,[r3+256] //Tail

str r1,[r4+256] //Head

moveSnake:

str r6,[r5+256] //Draw Apple each cycle, in case it is on snake

str r2,[r3+256] //Reset tail to Background

add r3,r3,#1 //Increment the tail pointer by 1

add r4,r4,#1 //Increment the head pointer by 1

str r1,[r4+256] //Draw new head

b moveSnake //Loop

# Complete code after Story 5

defineRegisters:

mov r1,#0x008844 //Snake colour (green)

mov r2,#0xffffff //Background colour (white)

mov r3, #271 //Tail position, initialised

mov r4, #272 //Head position, initialised

mov r5, #520 //Apple position

mov r6, #0xff8800 //Apple colour

drawSnake:

str r1,[r3+256] //Tail

str r1,[r4+256] //Head

moveSnake:

str r6,[r5+256] //Draw Apple each cycle, in case it is on snake

add r4,r4,#1 //Increment the head location by 1

cmp r4,r5 //If the head is in same location as apple...

beq moveHead //...Skip updating the tail, to make snake grow

moveTail:

str r2,[r3+256] //Reset tail to Background

add r3,r3,#1 //Increment the tail pointer by 1

moveHead:

str r1,[r4+256] //Draw new head

b moveSnake //Loop

# Complete code after Story 6

At the end of this story the code should be reverted to its state at the end of Story 5.

# Complete code after Story 7

defineRegisters:

mov r1,#0x008844 //Snake colour (green)

mov r2,#0xffffff //Background colour (white)

mov r3, #271 //Tail position, initialised

mov r4, #272 //Head position, initialised

mov r5, #520 //Apple position

mov r6, #0xff8800 //Apple colour

mov r7, #body //Pointer front of queue, initialised to first data loc

add r8,r7,#1 //Pointer to head address in body data (1 after tail)

InitialisePointers:

str r3, [r7] //r4 points to the tail address

str r4, [r8] //r3 points to the head address

drawSnake:

str r1,[r3+256] //Tail

str r1,[r4+256] //Head

moveSnake:

str r6,[r5+256] //Draw Apple each cycle, in case it is on snake

add r4,r4,#1 //Increment the head location by 1

cmp r4,r5 //If the head is in same location as apple...

beq moveHead //...Skip updating the tail, to make snake grow

moveTail:

ldr r0, [r7]

str r2,[r0+256] //Reset tail to Background

add r7,r7,#1 //Increment the tail pointer (for use next cycle)

moveHead:

add r8,r8,#1 //Increment the head pointer

str r4, [r8] //Store the new head location in data

str r1,[r4+256] //Draw new head

b moveSnake //Loop

body: dat 0 //Initial front of queue (screen address for tail)

# Complete code after Story 8

defineRegisters:

mov r1,#0x008844 //Snake colour (green)

mov r2,#0xffffff //Background colour (white)

mov r3, #271 //Tail position, initialised

mov r4, #272 //Head position, initialised

mov r5, #520 //Apple position

mov r6, #0xff8800 //Apple colour

mov r7, #body //Pointer front of queue, initialised to first data loc

add r8,r7,#1 //Pointer to head address in body data (1 after tail)

InitialisePointers:

str r3, [r7] //r4 points to the tail address

str r4, [r8] //r3 points to the head address

drawSnake:

str r1,[r3+256] //Tail

str r1,[r4+256] //Head

moveSnake:

str r6,[r5+256] //Draw Apple each cycle, in case it is on snake

add r4,r4,#1 //Increment the head location by 1

cmp r4,r5 //If the head is in same location as apple...

beq moveHead //...Skip updating the tail, to make snake grow

moveTail:

ldr r0, [r7]

str r2,[r0+256] //Reset tail to Background

add r7,r7,#1 //Increment the tail pointer (for use next cycle)

cmp r7,#200 //Check pointer is still within memory

blt moveHead

mov r7, #body //If not loop pointer back to start of body data

moveHead:

add r8,r8,#1 //Increment the head pointer

cmp r8,#200 //Check pointer is still within memory

blt updatePointer

mov r8, #body //If not loop pointer back to start of body data

updatePointer:

str r4, [r8] //Store the new head location in data

str r1,[r4+256] //Draw new head

b moveSnake //Loop

body: dat 0 //Initial front of queue (screen address for tail)

# Complete code after Story 9

defineRegisters:

mov r1,#0x008844 //Snake colour (green)

mov r2,#0xffffff //Background colour (white)

mov r3, #271 //Tail position, initialised

mov r4, #272 //Head position, initialised

mov r5, #520 //Apple position

mov r6, #0xff8800 //Apple colour

mov r7, #body //Pointer front of queue, initialised to first data loc

add r8,r7,#1 //Pointer to head address in body data (1 after tail)

mov r9, #0 //ASCII value of last key pressed

InitialisePointers:

str r3, [r7] //r4 points to the tail address

str r4, [r8] //r3 points to the head address

drawSnake:

str r1,[r3+256] //Tail

str r1,[r4+256] //Head

moveSnake:

str r6,[r5+256] //Draw Apple each cycle, in case it is on snake

inp r9,4 //Read the last key pressed (but do not wait for one)

cmp r9,#83 //S key

beq down

right:

add r4,r4,#1 //r9 holds the increment for the position...

b reDraw // Unconditional branch

down:

add r4,r4,#32 //...32 moves down one row on screen

reDraw:

cmp r4,r5 //If the head is in same location as apple...

beq moveHead //...Skip updating the tail, to make snake grow

moveTail:

ldr r0, [r7]

str r2,[r0+256] //Reset tail to Background

add r7,r7,#1 //Increment the tail pointer (for use next cycle)

cmp r7,#200 //Check pointer is still within memory

blt moveHead

mov r7, #body //If not loop pointer back to start of body data

moveHead:

add r8,r8,#1 //Increment the head pointer

cmp r8,#200 //Check pointer is still within memory

blt updatePointer

mov r8, #body //If not loop pointer back to start of body data

updatePointer:

str r4, [r8] //Store the new head location in data

str r1,[r4+256] //Draw new head

b moveSnake //Loop

body: dat 0 //Initial front of queue (screen address for tail)

# Complete code after Story 10

defineRegisters:

mov r1,#0x008844 //Snake colour (green)

mov r2,#0xffffff //Background colour (white)

mov r3, #271 //Tail position, initialised

mov r4, #272 //Head position, initialised

mov r5, #520 //Apple position

mov r6, #0xff8800 //Apple colour

mov r7, #body //Pointer front of queue, initialised to first data loc

add r8,r7,#1 //Pointer to head address in body data (1 after tail)

mov r9, #0 //ASCII value of last key pressed

InitialisePointers:

str r3, [r7] //r4 points to the tail address

str r4, [r8] //r3 points to the head address

drawSnake:

str r1,[r3+256] //Tail

str r1,[r4+256] //Head

moveSnake:

str r6,[r5+256] //Draw Apple each cycle, in case it is on snake

inp r9,4 //Read the last key pressed (but do not wait for one)

cmp r9,#87 //W key

beq up

cmp r9,#65 //A key

beq left

cmp r9,#83 //S key

beq down

cmp r9,#68 //D key

beq right

right:

add r4,r4,#1 //r9 holds the increment for the position...

b reDraw // Unconditional branch

down:

add r4,r4,#32 //...32 moves down one row on screen

b reDraw

up:

sub r4,r4,#32 //-32 moves up one row on screen

b reDraw

left:

sub r4,r4,#1 //-1 moves left

b reDraw

reDraw:

cmp r4,r5 //If the head is in same location as apple...

beq moveHead //...Skip updating the tail, to make snake grow

moveTail:

ldr r0, [r7]

str r2,[r0+256] //Reset tail to Background

add r7,r7,#1 //Increment the tail pointer (for use next cycle)

cmp r7,#200 //Check pointer is still within memory

blt moveHead

mov r7, #body //If not loop pointer back to start of body data

moveHead:

add r8,r8,#1 //Increment the head pointer

cmp r8,#200 //Check pointer is still within memory

blt updatePointer

mov r8, #body //If not loop pointer back to start of body data

updatePointer:

str r4, [r8] //Store the new head location in data

str r1,[r4+256] //Draw new head

b moveSnake //Loop

body: dat 0 //Initial front of queue (screen address for tail)

# Complete code after Story 11

defineRegisters:

mov r1,#0x008844 //Snake colour (green)

mov r2,#0xffffff //Background colour (white)

mov r3, #271 //Tail position, initialised

mov r4, #272 //Head position, initialised

mov r5, #0 //Apple position

mov r6, #0xff8800 //Apple colour

mov r7, #body //Pointer front of queue, initialised to first data loc

add r8,r7,#1 //Pointer to head address in body data (1 after tail)

mov r9, #0 //ASCII value of last key pressed

mov r10, #767 //Constant representing the size of screen memory

mov r11, #1023 //Constant

InitialisePointers:

str r3, [r7] //r4 points to the tail address

str r4, [r8] //r3 points to the head address

drawSnake:

str r1,[r3+256] //Tail

str r1,[r4+256] //Head

createApple:

inp r5,8 // gets a random 32 bit pattern

and r5,r5,r11 // r11 has 1023

cmp r5,r10 // r10 has 767

bgt createApple // restrict random range

cmp r5,r4 // Make sure apples is not located on head of snake

beq createApple

moveSnake:

str r6,[r5+256] //Draw Apple each cycle, in case it is on snake

inp r9,4 //Read the last key pressed (but do not wait for one)

cmp r9,#87 //W key

beq up

cmp r9,#65 //A key

beq left

cmp r9,#83 //S key

beq down

cmp r9,#68 //D key

beq right

right:

add r4,r4,#1 //r9 holds the increment for the position...

b reDraw // Unconditional branch

down:

add r4,r4,#32 //...32 moves down one row on screen

b reDraw

up:

sub r4,r4,#32 //-32 moves up one row on screen

b reDraw

left:

sub r4,r4,#1 //-1 moves left

b reDraw

reDraw:

cmp r4,r5 //If the head is in same location as apple...

beq moveHead //...Skip updating the tail, to make snake grow

moveTail:

ldr r0, [r7]

str r2,[r0+256] //Reset tail to Background

add r7,r7,#1 //Increment the tail pointer (for use next cycle)

cmp r7,#200 //Check pointer is still within memory

blt moveHead

mov r7, #body //If not loop pointer back to start of body data

moveHead:

add r8,r8,#1 //Increment the head pointer

cmp r8,#200 //Check pointer is still within memory

blt updatePointer

mov r8, #body //If not loop pointer back to start of body data

updatePointer:

str r4, [r8] //Store the new head location in data

str r1,[r4+256] //Draw new head

b moveSnake //Loop

body: dat 0 //Initial front of queue (screen address for tail)

# Complete code after Story 12

defineRegisters:

mov r1,#0x008844 //Snake colour (green)

mov r2,#0xffffff //Background colour (white)

mov r3, #271 //Tail position, initialised

mov r4, #272 //Head position, initialised

mov r5, #0 //Apple position

mov r6, #0xff8800 //Apple colour

mov r7, #body //Pointer front of queue, initialised to first data loc

add r8,r7,#1 //Pointer to head address in body data (1 after tail)

mov r9, #0 //ASCII value of last key pressed

mov r10, #767 //Constant representing the size of screen memory

mov r11, #1023 //Constant

InitialisePointers:

str r3, [r7] //r4 points to the tail address

str r4, [r8] //r3 points to the head address

drawSnake:

str r1,[r3+256] //Tail

str r1,[r4+256] //Head

createApple:

inp r5,8 // gets a random 32 bit pattern

and r5,r5,r11 // r11 has 1023

cmp r5,r10 // r10 has 767

bgt createApple // restrict random range

cmp r5,r4 // Make sure apples is not located on head of snake

beq createApple

moveSnake:

str r6,[r5+256] //Draw Apple each cycle, in case it is on snake

inp r9,4 //Read the last key pressed (but do not wait for one)

cmp r9,#87 //W key

beq up

cmp r9,#65 //A key

beq left

cmp r9,#83 //S key

beq down

cmp r9,#68 //D key

beq right

right:

add r4,r4,#1 //r9 holds the increment for the position...

b reDraw // Unconditional branch

down:

add r4,r4,#32 //...32 moves down one row on screen

b reDraw

up:

sub r4,r4,#32 //-32 moves up one row on screen

b reDraw

left:

sub r4,r4,#1 //-1 moves left

b reDraw

reDraw:

cmp r4,r5 //If the head is in same location as apple...

beq moveHead //...Skip updating the tail, to make snake grow

moveTail:

ldr r0, [r7]

str r2,[r0+256] //Reset tail to Background

add r7,r7,#1 //Increment the tail pointer (for use next cycle)

cmp r7,#200 //Check pointer is still within memory

blt moveHead

mov r7, #body //If not loop pointer back to start of body data

moveHead:

add r8,r8,#1 //Increment the head pointer

cmp r8,#200 //Check pointer is still within memory

blt updatePointer

mov r8, #body //If not loop pointer back to start of body data

updatePointer:

str r4, [r8] //Store the new head location in data

str r1,[r4+256] //Draw new head

cmp r4, r5 //Check again if the apple was eaten this cycle

beq createApple //If so, loop back to creating the apple

b moveSnake //Otherwise just repeat the move cycle

body: dat 0 //Initial front of queue (screen address for tail)

# Complete code after Story 13

defineRegisters:

mov r1,#0x008844 //Snake colour (green)

mov r2,#0xffffff //Background colour (white)

mov r3, #271 //Tail position, initialised

mov r4, #272 //Head position, initialised

mov r5, #0 //Apple position

mov r6, #0xff8800 //Apple colour

mov r7, #body //Pointer front of queue, initialised to first data loc

add r8,r7,#1 //Pointer to head address in body data (1 after tail)

mov r9, #0 //ASCII value of last key pressed

mov r10, #767 //Constant representing the size of screen memory

mov r11, #1023 //Constant

mov r12, #68 //Current direction of movement, initialised to 'right'

InitialisePointers:

str r3, [r7] //r4 points to the tail address

str r4, [r8] //r3 points to the head address

drawSnake:

str r1,[r3+256] //Tail

str r1,[r4+256] //Head

createApple:

inp r5,8 // gets a random 32 bit pattern

and r5,r5,r11 // r11 has 1023

cmp r5,r10 // r10 has 767

bgt createApple // restrict random range

cmp r5,r4 // Make sure apples is not located on head of snake

beq createApple

moveSnake:

str r6,[r5+256] //Draw Apple each cycle, in case it is on snake

inp r9,4 //Read the last key pressed (but do not wait for one)

switchOnKey:

cmp r9,#87 //W key

beq up

cmp r9,#65 //A key

beq left

cmp r9,#83 //S key

beq down

cmp r9,#68 //D key

beq right

mov r9, r12 //If not any of the recognised keys, use prev direction

b switchOnKey //and re-run the switch on key

right:

add r4,r4,#1 //r9 holds the increment for the position...

b reDraw // Unconditional branch

down:

add r4,r4,#32 //...32 moves down one row on screen

b reDraw

up:

sub r4,r4,#32 //-32 moves up one row on screen

b reDraw

left:

sub r4,r4,#1 //-1 moves left

b reDraw

reDraw:

mov r12,r9 //Update current direction with latest key

cmp r4,r5 //If the head is in same location as apple...

beq moveHead //...Skip updating the tail, to make snake grow

moveTail:

ldr r0, [r7]

str r2,[r0+256] //Reset tail to Background

add r7,r7,#1 //Increment the tail pointer (for use next cycle)

cmp r7,#200 //Check pointer is still within memory

blt moveHead

mov r7, #body //If not loop pointer back to start of body data

moveHead:

add r8,r8,#1 //Increment the head pointer

cmp r8,#200 //Check pointer is still within memory

blt updatePointer

mov r8, #body //If not loop pointer back to start of body data

updatePointer:

str r4, [r8] //Store the new head location in data

str r1,[r4+256] //Draw new head

cmp r4, r5 //Check again if the apple was eaten this cycle

beq createApple //If so, loop back to creating the apple

b moveSnake //Otherwise just repeat the move cycle

body: dat 0 //Initial front of queue (screen address for tail)

# Complete code after Story 14

defineRegisters:

mov r1,#0x008844 //Snake colour (green)

mov r2,#0xffffff //Background colour (white)

mov r3, #271 //Tail position, initialised

mov r4, #272 //Head position, initialised

mov r5, #0 //Apple position

mov r6, #0xff8800 //Apple colour

mov r7, #body //Pointer front of queue, initialised to first data loc

add r8,r7,#1 //Pointer to head address in body data (1 after tail)

mov r9, #0 //ASCII value of last key pressed

mov r10, #767 //Constant representing the size of screen memory

mov r11, #1023 //Constant

mov r12, #68 //Current direction of movement, initialised to 'right'

InitialisePointers:

str r3, [r7] //r4 points to the tail address

str r4, [r8] //r3 points to the head address

drawSnake:

str r1,[r3+256] //Tail

str r1,[r4+256] //Head

createApple:

inp r5,8 // gets a random 32 bit pattern

and r5,r5,r11 // r11 has 1023

cmp r5,r10 // r10 has 767

bgt createApple // restrict random range

cmp r5,r4 // Make sure apples is not located on head of snake

beq createApple

moveSnake:

str r6,[r5+256] //Draw Apple each cycle, in case it is on snake

inp r9,4 //Read the last key pressed (but do not wait for one)

switchOnKey:

cmp r9,#87 //W key

beq up

cmp r9,#65 //A key

beq left

cmp r9,#83 //S key

beq down

cmp r9,#68 //D key

beq right

mov r9, r12 //If not any of the recognised keys, use prev direction

b switchOnKey //and re-run the switch on key

right:

add r4,r4,#1 //Adding 1 to location moves right

and r0,r4,#31

cmp r0,#0

beq gameOver

b reDraw

down:

add r4,r4,#32 //32 moves down one row on screen

cmp r4,r10

bgt gameOver

b reDraw

up:

sub r4,r4,#32 //-32 moves up one row on screen

cmp r4,#0

blt gameOver

b reDraw

left:

sub r4,r4,#1 //-1 moves left

and r0,r4,#31

cmp r0,#31

beq gameOver

b reDraw

reDraw:

mov r12,r9 //Update current direction with latest key

cmp r4,r5 //If the head is in same location as apple...

beq moveHead //...Skip updating the tail, to make snake grow

moveTail:

ldr r0, [r7]

str r2,[r0+256] //Reset tail to Background

add r7,r7,#1 //Increment the tail pointer (for use next cycle)

cmp r7,#200 //Check pointer is still within memory

blt moveHead

mov r7, #body //If not loop pointer back to start of body data

moveHead:

add r8,r8,#1 //Increment the head pointer

cmp r8,#200 //Check pointer is still within memory

blt updatePointer

mov r8, #body //If not loop pointer back to start of body data

updatePointer:

str r4, [r8] //Store the new head location in data

str r1,[r4+256] //Draw new head

cmp r4, r5 //Check again if the apple was eaten this cycle

beq createApple //If so, loop back to creating the apple

b moveSnake //Otherwise just repeat the move cycle

gameOver:

halt //To stop program execution running into data area

body: dat 0 //Initial front of queue (screen address for tail)

# Complete code after Story 15

defineRegisters:

mov r1,#0x008844 //Snake colour (green)

mov r2,#0xffffff //Background colour (white)

mov r3, #271 //Tail position, initialised

mov r4, #272 //Head position, initialised

mov r5, #0 //Apple position

mov r6, #0xff8800 //Apple colour

mov r7, #body //Pointer front of queue, initialised to first data loc

add r8,r7,#1 //Pointer to head address in body data (1 after tail)

mov r9, #0 //ASCII value of last key pressed

mov r10, #767 //Constant representing the size of screen memory

mov r11, #1023 //Constant

mov r12, #68 //Current direction of movement, initialised to 'right'

InitialisePointers:

str r3, [r7] //r4 points to the tail address

str r4, [r8] //r3 points to the head address

drawSnake:

str r1,[r3+256] //Tail

str r1,[r4+256] //Head

createApple:

inp r5,8 // gets a random 32 bit pattern

and r5,r5,r11 // r11 has 1023

cmp r5,r10 // r10 has 767

bgt createApple // restrict random range

cmp r5,r4 // Make sure apples is not located on head of snake

beq createApple

moveSnake:

str r6,[r5+256] //Draw Apple each cycle, in case it is on snake

inp r9,4 //Read the last key pressed (but do not wait for one)

switchOnKey:

cmp r9,#87 //W key

beq up

cmp r9,#65 //A key

beq left

cmp r9,#83 //S key

beq down

cmp r9,#68 //D key

beq right

dontChangeDirection:

mov r9, r12 //If not any of the recognised keys, use prev direction

b switchOnKey //and re-run the switch on key

right:

cmp r12,#65

beq dontChangeDirection

add r4,r4,#1 //Adding 1 to location moves right

and r0,r4,#31

cmp r0,#0

beq gameOver

b reDraw

down:

cmp r12,#87

beq dontChangeDirection

add r4,r4,#32 //32 moves down one row on screen

cmp r4,r10

bgt gameOver

b reDraw

up:

cmp r12,#83

beq dontChangeDirection

sub r4,r4,#32 //-32 moves up one row on screen

cmp r4,#0

blt gameOver

b reDraw

left:

cmp r12,#68

beq dontChangeDirection

sub r4,r4,#1 //-1 moves left

and r0,r4,#31

cmp r0,#31

beq gameOver

b reDraw

reDraw:

mov r12,r9 //Update current direction with latest key

cmp r4,r5 //If the head is in same location as apple...

beq moveHead //...Skip updating the tail, to make snake grow

moveTail:

ldr r0, [r7]

str r2,[r0+256] //Reset tail to Background

add r7,r7,#1 //Increment the tail pointer (for use next cycle)

cmp r7,#200 //Check pointer is still within memory

blt moveHead

mov r7, #body //If not loop pointer back to start of body data

moveHead:

add r8,r8,#1 //Increment the head pointer

cmp r8,#200 //Check pointer is still within memory

blt updatePointer

mov r8, #body //If not loop pointer back to start of body data

updatePointer:

str r4, [r8] //Store the new head location in data

str r1,[r4+256] //Draw new head

cmp r4, r5 //Check again if the apple was eaten this cycle

beq createApple //If so, loop back to creating the apple

b moveSnake //Otherwise just repeat the move cycle

gameOver:

halt //To stop program execution running into data area

body: dat 0 //Initial front of queue (screen address for tail)

# Complete code after Story 16

defineRegisters:

mov r1,#0x008844 //Snake colour (green)

mov r2,#0xffffff //Background colour (white)

mov r3, #271 //Tail position, initialised

mov r4, #272 //Head position, initialised

mov r5, #0 //Apple position

mov r6, #0xff8800 //Apple colour

mov r7, #body //Pointer front of queue, initialised to first data loc

add r8,r7,#1 //Pointer to head address in body data (1 after tail)

mov r9, #0 //ASCII value of last key pressed

mov r10, #767 //Constant representing the size of screen memory

mov r11, #1023 //Constant

mov r12, #68 //Current direction of movement, initialised to 'right'

InitialisePointers:

str r3, [r7] //r4 points to the tail address

str r4, [r8] //r3 points to the head address

drawSnake:

str r1,[r3+256] //Tail

str r1,[r4+256] //Head

createApple:

inp r5,8 // gets a random 32 bit pattern

and r5,r5,r11 // r11 has 1023

cmp r5,r10 // r10 has 767

bgt createApple // restrict random range

cmp r5,r4 // Make sure apples is not located on head of snake

beq createApple

moveSnake:

str r6,[r5+256] //Draw Apple each cycle, in case it is on snake

inp r9,4 //Read the last key pressed (but do not wait for one)

switchOnKey:

cmp r9,#87 //W key

beq up

cmp r9,#65 //A key

beq left

cmp r9,#83 //S key

beq down

cmp r9,#68 //D key

beq right

dontChangeDirection:

mov r9, r12 //If not any of the recognised keys, use prev direction

b switchOnKey //and re-run the switch on key

right:

cmp r12,#65

beq dontChangeDirection

add r4,r4,#1 //Adding 1 to location moves right

and r0,r4,#31

cmp r0,#0

beq gameOver

b reDraw

down:

cmp r12,#87

beq dontChangeDirection

add r4,r4,#32 //32 moves down one row on screen

cmp r4,r10

bgt gameOver

b reDraw

up:

cmp r12,#83

beq dontChangeDirection

sub r4,r4,#32 //-32 moves up one row on screen

cmp r4,#0

blt gameOver

b reDraw

left:

cmp r12,#68

beq dontChangeDirection

sub r4,r4,#1 //-1 moves left

and r0,r4,#31

cmp r0,#31

beq gameOver

b reDraw

reDraw:

mov r12,r9 //Update current direction with latest key

cmp r4,r5 //If the head is in same location as apple...

beq moveHead //...Skip updating the tail, to make snake grow

moveTail:

ldr r0, [r7]

str r2,[r0+256] //Reset tail to Background

add r7,r7,#1 //Increment the tail pointer (for use next cycle)

cmp r7,#200 //Check pointer is still within memory

blt moveHead

mov r7, #body //If not loop pointer back to start of body data

moveHead:

add r8,r8,#1 //Increment the head pointer

cmp r8,#200 //Check pointer is still within memory

blt updatePointer

mov r8, #body //If not loop pointer back to start of body data

updatePointer:

str r4, [r8] //Store the new head location in data

checkForCrossing: // check if the snake has hit itself

ldr r0,[r4+256] // read, from screen, colour of pixel head is moving to

cmp r0,r1 //If it is snake colour

beq gameOver

str r1,[r4+256] //Draw new head

cmp r4, r5 //Check again if the apple was eaten this cycle

beq createApple //If so, loop back to creating the apple

b moveSnake //Otherwise just repeat the move cycle

gameOver:

halt //To stop program execution running into data area

body: dat 0 //Initial front of queue (screen address for tail)

# Complete code after Story 17

defineRegisters:

mov r1,#0x008844 //Snake colour (green)

mov r2,#0xffffff //Background colour (white)

mov r3, #271 //Tail position, initialised

mov r4, #272 //Head position, initialised

mov r5, #0 //Apple position

mov r6, #0xff8800 //Apple colour

mov r7, #body //Pointer front of queue, initialised to first data loc

add r8,r7,#1 //Pointer to head address in body data (1 after tail)

mov r9, #0 //ASCII value of last key pressed

mov r10, #767 //Constant representing the size of screen memory

mov r11, #1023 //Constant

mov r12, #68 //Current direction of movement, initialised to 'right'

InitialisePointers:

str r3, [r7] //r4 points to the tail address

str r4, [r8] //r3 points to the head address

drawSnake:

str r1,[r3+256] //Tail

str r1,[r4+256] //Head

createApple:

inp r5,8 // gets a random 32 bit pattern

and r5,r5,r11 // r11 has 1023

cmp r5,r10 // r10 has 767

bgt createApple // restrict random range

cmp r5,r4 // Make sure apples is not located on head of snake

beq createApple

moveSnake:

str r6,[r5+256] //Draw Apple each cycle, in case it is on snake

inp r9,4 //Read the last key pressed (but do not wait for one)

switchOnKey:

cmp r9,#87 //W key

beq up

cmp r9,#65 //A key

beq left

cmp r9,#83 //S key

beq down

cmp r9,#68 //D key

beq right

dontChangeDirection:

mov r9, r12 //If not any of the recognised keys, use prev direction

b switchOnKey //and re-run the switch on key

right:

cmp r12,#65

beq dontChangeDirection

add r4,r4,#1 //Adding 1 to location moves right

and r0,r4,#31

cmp r0,#0

beq gameOver

b reDraw

down:

cmp r12,#87

beq dontChangeDirection

add r4,r4,#32 //32 moves down one row on screen

cmp r4,r10

bgt gameOver

b reDraw

up:

cmp r12,#83

beq dontChangeDirection

sub r4,r4,#32 //-32 moves up one row on screen

cmp r4,#0

blt gameOver

b reDraw

left:

cmp r12,#68

beq dontChangeDirection

sub r4,r4,#1 //-1 moves left

and r0,r4,#31

cmp r0,#31

beq gameOver

b reDraw

reDraw:

mov r12,r9 //Update current direction with latest key

cmp r4,r5 //If the head is in same location as apple...

beq moveHead //...Skip updating the tail, to make snake grow

moveTail:

ldr r0, [r7]

str r2,[r0+256] //Reset tail to Background

add r7,r7,#1 //Increment the tail pointer (for use next cycle)

cmp r7,#200 //Check pointer is still within memory

blt moveHead

mov r7, #body //If not loop pointer back to start of body data

moveHead:

add r8,r8,#1 //Increment the head pointer

cmp r8,#200 //Check pointer is still within memory

blt updatePointer

mov r8, #body //If not loop pointer back to start of body data

updatePointer:

str r4, [r8] //Store the new head location in data

checkForCrossing: // check if the snake has hit itself

ldr r0,[r4+256] // read, from screen, colour of pixel head is moving to

cmp r0,r1 //If it is snake colour

beq gameOver

checkForMaxLength:

cmp r7,r8 //If the front has caught up with rear, then body data full

beq gameWin

str r1,[r4+256] //Draw new head

cmp r4, r5 //Check again if the apple was eaten this cycle

beq createApple //If so, loop back to creating the apple

b moveSnake /Otherwise just repeat the move cycle

gameWin:

mov r0, #message

out r0,8 //Write message as text into output window

gameOver:

halt //To stop program execution running into data area

message: //ASCII encoding of 'You win' message (in reverse)

dat 0x20756f59 //' uoY'

dat 0x006e6977 //'niw'

body: dat 0 //Initial front of queue (screen address for tail)