AQA

AS
COMPUTER SCIENCE

Paper 1

June 2023

Preliminary Material

To be opened and issued to candidates on or after **1 March 2023** subject to the instructions given in the **Teachers' Notes** (7516/1/TN).

Note

- The **Preliminary Material**, **Skeleton Program** and **Data Files** are to be seen by candidates and their teachers **only**, for use during preparation for the examination on **Tuesday 16 May 2023**. They **cannot** be used by anyone else for any other purpose, other than that stated in the instructions issued, until after the examination date has passed. They must **not** be provided to third parties.

Information

- A Skeleton Program is provided separately by your teacher and must be read in conjunction with this Preliminary Material.
- You are advised to familiarise yourself with the Preliminary Material and Skeleton Program before the examination.
- A copy of this Preliminary Material and the Skeleton Program will be made available to you in hard copy and electronically at the start of the examination.
- You must **not** take any copy of the Preliminary Material, Skeleton Program and Data Files or any other material into the examination room.

Candidates will need access to a text file editor, such as Notepad or TextEdit.

INSTRUCTIONS FOR CANDIDATES

The question paper is divided into **three** sections.

Section A

You will be asked to create a new program and answer questions **not** related to the **Preliminary Material** or **Skeleton Program**.

Section B

Questions will refer to the **Preliminary Material** and the **Skeleton Program**, but will not require programming.

Section C

Questions will use the **Preliminary Material** and the **Skeleton Program** and may require the `prog1.txt`, `prog2.txt` and `prog3.txt` **Data Files**.

Electronic Answer Document

Answers for **all** questions, for **all** sections, must be entered into the word-processed document made available to you at the start of the examination and referred to in the question paper rubrics as the **Electronic Answer Document**.

Preparation for the Examination

You should ensure that you are familiar with this **Preliminary Material** and the **Skeleton Program** for your programming language.

Assembler Simulator

The **Skeleton Program** accompanying this **Preliminary Material** is a simple assembler simulator.

> The processor model and assembly language used in this Preliminary Material are very different to that used in AQA AS Computer Science Paper 2. This Preliminary Material is **not** intended to be used to learn about the standard AQA assembly language.

The simulator is based on a simple processor with a single general-purpose register called the accumulator (`ACC`).

Other registers in this simulation include the program counter (`PC`), the status register (`STATUS`), the stack pointer (`TOS`) and a register for flagging runtime errors (`ERR`).

The status register has three flags: `Z`, `N` and `V`

The `Z`, `N` and `V` flags are set after an instruction is executed.

The `Z` flag is set to 1 if the result of executing the instruction is the number zero. Otherwise the flag is cleared (set to 0).

The `N` flag is set to 1 if the result of executing the instruction is a negative value. Otherwise the flag is cleared (set to 0).

The `V` flag is set to 1 if the result of executing the instruction is a value that cannot be represented in 8 bits. Otherwise the flag is cleared (set to 0).

The format of an instruction is:

```
<LABEL>: <OPCODE> <OPERAND><COMMENT>
```

Table 1 explains the format for the different components of an instruction. Some of the components are optional for some instructions.

Table 1

Instruction component	Explanation
`<LABEL>`	Up to 5 characters in length followed by a colon. If the label is shorter than 5 characters, then spaces should be inserted so that the colon is always the sixth character in the instruction. The colon is followed by a space.
`<OPCODE>`	Up to 4 characters in length, followed by a space
`<OPERAND>`	Of variable length
`<COMMENT>`	Of variable length, starts with the character `*`

Table 2 explains aspects of the Assembler Simulator instruction set.

Table 2

Opcode	Explanation
LDA	Loads the accumulator with the value in the memory location specified by the address in the operand.
STA	Stores the contents of the accumulator in the memory location specified by the address in the operand.
LDA#	Loads the accumulator with the operand value.
SKP	Skips to the next instruction.
ADD	Adds the value stored in the memory location specified by the address in the operand to the value in the accumulator.
SUB	Subtracts the value stored in the memory location specified by the address in the operand from the value in the accumulator.
JMP	Jumps to the instruction in the memory location specified by the address in the operand.
CMP#	Compares the value in the accumulator with the operand value and sets the status register flags accordingly. The Z flag is set to 1 when the contents of the accumulator are equal to the operand value, otherwise the Z flag is set to 0 The N flag is set to 1 when the contents of the accumulator are less than the operand value, otherwise the N flag is set to 0
BEQ	Jumps to the instruction in the memory location specified by the address in the operand if the Z flag is set (1).
JSR	Jumps to the subroutine starting at the memory location specified by the address in the operand. The return address is stored on the top of the stack. The top of the stack pointer is updated and the contents of the program counter are stored on the top of the stack. (See **Figure 4,** on page 9)
RTN	Returns from the current subroutine. The top of the stack stores the return address. The return address is copied to the program counter. The top of the stack pointer is updated. (See **Figure 4,** on page 9)
HLT	Stops execution of the program.

The assembly language used in this simulator uses symbolic addresses. A symbolic address is an alphanumeric label referencing a memory location.

For example, in **Figure 1**, the first line (`NUM1:  2`) has the label `NUM1` which is the symbolic address for memory location 1

The fifth line (`START: LDA  NUM1`) uses the symbolic address `NUM1` as the operand.

Figure 1 shows the assembly language source code from the `prog1.txt` data file.

Figure 1

```
 NUM1:      2
 NUM2:      5
 NUM3:      -1
 NUM4:      125
START: LDA  NUM1    * test while loop
WHILE: CMP# 12
       BEQ  WEND
       ADD  NUM2
       JMP  WHILE
       SKP
 WEND: STA  NUM3
       ADD  NUM4
       HLT
```

Figure 2 shows the assembly language source code from the `prog2.txt` data file.

Figure 2

```
       LDA# 3       * test negative
       SUB  NUM1
       SKP
       STA  FINAL
       HLT




 NUM1:      5
FINAL:      0
```

The source code is loaded from a text file and symbolic addresses (labels) are converted into memory addresses during the assembly process.

The resulting code is interpreted by the simulator and the output shows the original program alongside the assembled code.

The memory contents after loading and assembling the program are shown in `Frame 0`. Each subsequent frame shows the result of executing one instruction.

For example, the output shown in **Figure 3** is produced by assembling and running `prog2.txt`

Figure 3

```
****** Frame 0 ************************************************
*
*  Memory     Location  Label  Op   Operand Comment
*  Contents                    Code
*  JMP  1     |   0  |
*  LDA# 3     |   1  |         LDA# 3       * test negative
*  SUB  10    |   2  |         SUB  NUM1
*  SKP  0     |   3  |         SKP
*  STA  11    |   4  |         STA  FINAL
*  HLT  0     |   5  |         HLT
*       0     |   6  |
*       0     |   7  |
*       0     |   8  |
*       0     |   9  |
*       5     |  10  |   NUM1:      5
*       0     |  11  |  FINAL:      0
*
*  PC:  0  ACC:  0  TOS:  20
*  Status Register: ZNV
*                   100
************************************************************


****** Frame 1 ************************************************
*  Current Instruction Register:  JMP 1
*
*  Memory     Location  Label  Op   Operand Comment
*  Contents                    Code
*  JMP  1     |   0  |
*  LDA# 3     |   1  |         LDA# 3       * test negative
*  SUB  10    |   2  |         SUB  NUM1
*  SKP  0     |   3  |         SKP
*  STA  11    |   4  |         STA  FINAL
*  HLT  0     |   5  |         HLT
*       0     |   6  |
*       0     |   7  |
*       0     |   8  |
*       0     |   9  |
*       5     |  10  |   NUM1:      5
*       0     |  11  |  FINAL:      0
*
*  PC:  1  ACC:  0  TOS:  20
*  Status Register: ZNV
*                   100
************************************************************
```

```
****** Frame 2 ************************************************
*  Current Instruction Register:  LDA# 3
*
*  Memory     Location  Label  Op   Operand Comment
*  Contents                    Code
*  JMP  1     |   0  |
*  LDA# 3     |   1  |         LDA# 3       * test negative
*  SUB  10    |   2  |         SUB  NUM1
*  SKP  0     |   3  |         SKP
*  STA  11    |   4  |         STA  FINAL
*  HLT  0     |   5  |         HLT
*       0     |   6  |
*       0     |   7  |
*       0     |   8  |
*       0     |   9  |
*       5     |  10  |   NUM1:      5
*       0     |  11  |  FINAL:      0
*
*  PC:  2  ACC:  3  TOS:  20
*  Status Register: ZNV
*                   000
************************************************************


****** Frame 3 ************************************************
*  Current Instruction Register:  SUB 10
*
*  Memory     Location  Label  Op   Operand Comment
*  Contents                    Code
*  JMP  1     |   0  |
*  LDA# 3     |   1  |         LDA# 3       * test negative
*  SUB  10    |   2  |         SUB  NUM1
*  SKP  0     |   3  |         SKP
*  STA  11    |   4  |         STA  FINAL
*  HLT  0     |   5  |         HLT
*       0     |   6  |
*       0     |   7  |
*       0     |   8  |
*       0     |   9  |
*       5     |  10  |   NUM1:      5
*       0     |  11  |  FINAL:      0
*
*  PC:  3  ACC:  -2  TOS:  20
*  Status Register: ZNV
*                   010
************************************************************
```

```
****** Frame 4 ************************************************
*  Current Instruction Register:  SKP 0
*
*  Memory     Location  Label  Op   Operand Comment
*  Contents                    Code
*  JMP  1     |   0  |
*  LDA# 3     |   1  |         LDA# 3       * test negative
*  SUB  10    |   2  |         SUB  NUM1
*  SKP  0     |   3  |         SKP
*  STA  11    |   4  |         STA  FINAL
*  HLT  0     |   5  |         HLT
*       0     |   6  |
*       0     |   7  |
*       0     |   8  |
*       0     |   9  |
*       5     |  10  |   NUM1:      5
*       0     |  11  |  FINAL:      0
*
*  PC:  4  ACC:  -2  TOS:  20
*  Status Register: ZNV
*                   010
************************************************************


****** Frame 5 ************************************************
*  Current Instruction Register:  STA 11
*
*  Memory     Location  Label  Op   Operand Comment
*  Contents                    Code
*  JMP  1     |   0  |
*  LDA# 3     |   1  |         LDA# 3       * test negative
*  SUB  10    |   2  |         SUB  NUM1
*  SKP  0     |   3  |         SKP
*  STA  11    |   4  |         STA  FINAL
*  HLT  0     |   5  |         HLT
*       0     |   6  |
*       0     |   7  |
*       0     |   8  |
*       0     |   9  |
*       5     |  10  |   NUM1:      5
*       -2    |  11  |  FINAL:      0
*
*  PC:  5  ACC:  -2  TOS:  20
*  Status Register: ZNV
*                   010
************************************************************
Execution terminated
```

A stack is used to store the return address when a subroutine is called. In this simulator, the stack is built downwards from the highest memory location. The `TOS` register stores the address of the most recent return address pushed onto the stack. Before a return address is stored on the stack, the `TOS` is decremented by 1. After a return address is retrieved from the stack, the `TOS` is incremented by 1.

The example in **Figure 4** shows the memory contents after the instruction in location 2 has been executed. The return address (location 3) has been stored at the top of the stack in memory location 18 (`HI_MEM`, the highest memory location, is 19). Note this is not the output produced by the **Skeleton Program**.

When the instruction `RTN` in location 5 is executed, the return address (location 3) is copied from the top of the stack (location 18) into the program counter. The contents of `TOS` is then increased to point to location 19. See **Figure 5,** on page 10.

Figure 4

```
Memory      Location  Label   Op    Operand
Contents                      Code
JMP  6      |   0   |
ADD  10     |   1   |    SUB1: ADD   NUM1
JSR  4      |   2   |          JSR   SUB2
RTN  0      |   3   |          RTN
ADD  11     |   4   |    SUB2: ADD   NUM2
RTN  0      |   5   |          RTN
LDA  10     |   6   |  START:  LDA   NUM1
JSR  1      |   7   |          JSR   SUB1
STA  12     |   8   |          STA   TOTAL
HLT  0      |   9   |          HLT
     7      |  10   |    NUM1:       7
     0      |  11   |    NUM2:       0
     0      |  12   |  TOTAL:        0
     0      |  13   |
     0      |  14   |
     0      |  15   |
     0      |  16   |
     0      |  17   |
     3      |  18   |
     8      |  19   |

PC:  4  ACC:  14  TOS:  18
```

Figure 5

```
Memory      Location  Label   Op    Operand
Contents                      Code
JMP  6      |   0   |
ADD  10     |   1   |    SUB1: ADD   NUM1
JSR  4      |   2   |          JSR   SUB2
RTN  0      |   3   |          RTN
ADD  11     |   4   |    SUB2: ADD   NUM2
RTN  0      |   5   |          RTN
LDA  10     |   6   |   START: LDA   NUM1
JSR  1      |   7   |          JSR   SUB1
STA  12     |   8   |          STA   TOTAL
HLT  0      |   9   |          HLT
     7      |  10   |    NUM1:       7
     0      |  11   |    NUM2:       0
     0      |  12   |   TOTAL:       0
     0      |  13   |
     0      |  14   |
     0      |  15   |
     0      |  16   |
     0      |  17   |
     3      |  18   |
     8      |  19   |

PC:  3  ACC:  14  TOS:  19
```

Figure 6 shows the source code from the `prog3.txt` data file.

Figure 6

```
        SUB1: ADD  NUM1    * test subroutines
              JSR  SUB2
              RTN
        SUB2: ADD  NUM1
              JSR  SUB3
              RTN
        SUB3: ADD  NUM1
              JSR  SUB4
              RTN
        SUB4: ADD  NUM1
              RTN
       START: LDA  NUM1
              JSR  SUB1
              STA  NUM2
              HLT
        NUM1:      7
        NUM2:      0
```

END OF PRELIMINARY MATERIAL

There is no preliminary material on this page

There is no preliminary material on this page