BSc, Beng and MEng Degree Examinations 2021–22

DEPARTMENT OF COMPUTER SCIENCE

System and Devices 1

Open Assessment

Issued:	Wednesday, 20th April, 2022 Midday
Submission due:	Wednesday, 11th May, 2022 Midday
Feedback and Marks due:	Wednesday, 8th June, 2022

All students should submit their answers through the electronic submission system: http://www.cs.york.ac.uk/student/assessment/submit/

An assessment that has been submitted after this deadline will be marked initially as if it had been handed in on time, but the Board of Examiners will normally apply a lateness penalty.

Your attention is drawn to the section about Academic Misconduct in your Departmental Handbook: https://www.cs.york.ac.uk/student/handbook/.

All queries on this assessment should be drawn to:

- Dr Mike Freeman (mjf@cs.york.ac.uk).

Answers that apply to all students will be posted on the VLE.

SYS1 Open Assessment 2021/22

This assessment continues the work started in practical 9. When answering these questions you must use the assembler and ISE project files from the Exam folder on the VLE.

The bug trap's functionality and accuracy are to be further improved by the addition of a camera. To implement the required bug detection algorithms a number of new software routines are required. Your task is to implement these functions. Where required new instructions and additional hardware may be added to improve processing performance. However, to ensure compatibility with other systems the following restrictions are imposed:

- The maximum clock speed used by the system is limited to 10MHz.
- Instruction names and machine code implementations i.e. bit fields used, are restricted to those defined in Appendix A.
- The functionality and hardware of existing instructions may not be altered e.g. the rotate left, adder/subtractor and bitwise-logic hardware within the ALU may not be changed.
- Existing schematic symbols e.g. `alu.sch` etc, may not be altered. However, you may add additional processing elements to the `ALU` schematic to enable new functionality, but, you may not alter its ports i.e. its interface. To control this new hardware the control logic : `decode.vhd`, may also be updated. All other schematics must not be modified.
- New schematics may not use existing Xilinx library arithmetic components, VHDL components, or IP core generated components.
- The assembler `simpleCPUv1d_as.py` may not be modified.
- Only a single macro file : `simpleCPUv1d.m4` may be used.
- Input images are limited to the file : `bug24x24.ppm` which can be downloaded from the Exam folder on the VLE.
- Output images are limited to 24×24 pixels in size and must use the PPM or PGM image formats, as defined in each question.

If a solution breaks any of these restrictions you will be awarded a zero mark for the questions affected.

You may implement any of the instructions defined in Appendix A. However, you do not need to do so to complete the programming tasks in questions 1, 2 or 4. There are five undefined instructions that you may add to your system to improve processing performance:

- XOP1 (undefined)
- XOP2 (register indirect)
- XOP3 (register)
- XOP4 (register indirect)
- XOP5 (register)

The assembler has already been updated to support these new instructions and may not be altered. The addressing mode of XOP1 is undefined i.e. it can use any addressing mode that is compatible with the assembler. The addressing modes of instructions XOP2 – XOP5 are fixed and can not be changed. The functions performed by each of these instructions can be changed for each question e.g. XOP5 could implement a register addressing mode ADD instruction in one question and a register addressing mode SUB instruction in another.

At the end of each question there is a description of the files that you need to submit. These files should be placed in a directory of the same name i.e. `Q1`, `Q2`, `Q3`, or `Q4`. When complete these directories should be compressed into a single file: `SYS1.zip` and uploaded through the submission system.

Q1 (30 marks)

Square : write a program to draw a square in the RGB image stored in memory. The position and size of the square is fixed. The top left corner is co-ordinate (4,4), each side is 16 pixels long, alternating in colour between yellow and purple, as shown in figure 1. This image is stored in memory using the packed RGB data type shown in figure 2. For more information on this data type refer to practical 9. This program must be named : `square.asm` and submitted as a plain text file.

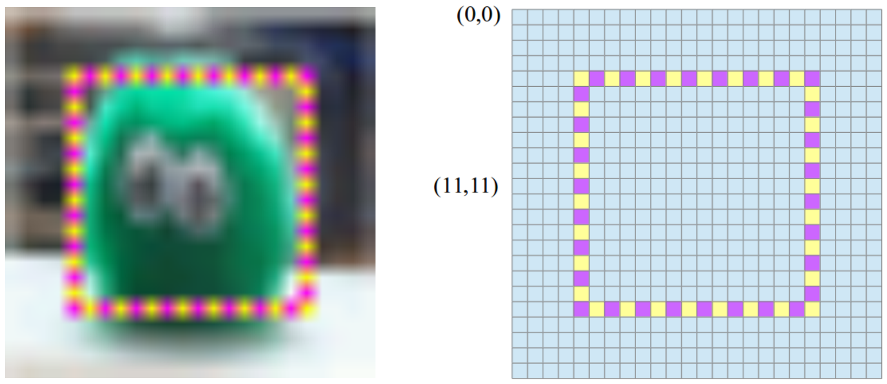

Figure 1 : draw box subroutine output.

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
7	6	5	4	3	7	6	5	4	3	2	7	6	5	4	3

Figure 2 : 16-bit packed RGB data format

The memory model `ram_4Kx16_sim` has already been configured to automatically load the test image: `bug24x24.ppm` and store it at base address 1024. When your program has finished processing this image it must store the value 0xFF to address 0xFFF to trigger the generation of the output image file: `output.ppm`.

Marks are awarded for:

- Functionality (14 marks) : correctly drawing the square.
- Processing performance (8 marks) : how quickly your program performs this task. Marks will be deducted for incorrect or missing RTL descriptions of any new instructions used.
- Minimising program size (8 marks) : minimising storage i.e. instructions and program data. Image data is not included in this calculation.

Submission: for this question you should submit:

- `square.asm` : a plain text file, assembly language program used to implement the square functionality.
- You may submit a single macro file : `simpleCPUv1d.m4`. If macros are used you must submit a plain text file : `build.txt`, containing clear instructions of how this file will be used in the build process.
- Any ISE schematics (`.sch` files) that have been modified to implement this system. **Note**, remember the schematic restrictions defined on page 2.
- Any new ISE schematics (`.sch` files) and symbols (`.sym` files) that have been created to implement this system.

- The file `decoder.vhd` if you have implement any new instructions. You must also submit a plain text file : `instructions.txt`, containing the RTL descriptions of the micro-instructions used to implement these new instructions during the Fetch, Decode and Execute phases, and a brief describe of their operation.

These files will be added to the base ISE project from the Exam folder on the VLE for testing. Do **<u>not</u>** submit the full ISE project, or the output image. The output image will be regenerated during the marking process. Files that do not comply to the stated file formats will be awarded a zero mark.

Q2 (30 marks)

Encrypt : write a program using the base instruction-set to encrypt an RGB image using a simple block cipher i.e. you are not allowed to add new instructions to your system when implementing this solution. Encryption is performed using a Feistel cipher as shown in figure 3.

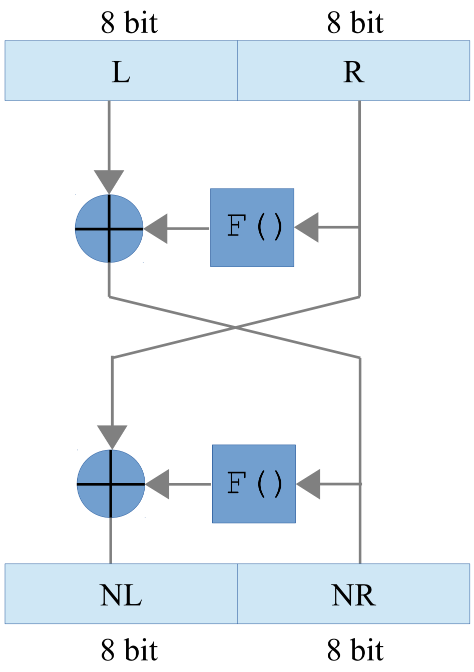

Figure 3 : Feistel cipher

Each 16bit value stored in memory is split into left (L) and right (R) bytes. These are combined using an encryption function `F()` and the bitwise XOR function ⊕, to produce the new left (NL) and right (NR) bytes. The encryption function `F()` used is a simple bit reverse, as illustrated in figure 4.

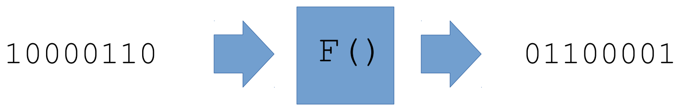

Figure 4 : bit reverse

This program must be named: `encrypt.asm` and submitted as a plain text file. Again, the memory model `ram_4Kx16_sim` has already been configured to automatically load the test image: `bug24x24.ppm` and store it at base address 1024. When your program has finished processing this image it must store the value 0xFF to address 0xFFF to trigger the generation of the output image file: `output.ppm`.

Marks are awarded for:

- Functionality (14 marks) : correctly encrypting the image data.
- Processing performance (8 marks) : how quickly your program performs this task.
- Minimising program size (8 marks) : minimising storage i.e. instructions and program data. Image data is not included in this calculation.

To test if your encryption program has worked correctly the python program `decrypt.py` can be downloaded from the VLE. This will decrypt the encrypted PPM image `output.ppm` to produce a new file call `new.ppm`.

```
python decrypt.py -i output
```

Note, this new image will be comparable to the original i.e. original and unencrypted images will look the same, but there will be some small rounding errors in the unencrypted image caused during the conversion to the packed RGB data type.

Submission: for this question you should submit:

- `encrypt.asm` : a plain text file, assembly language program used to implement the encryption functionality.
- You may submit a single macro file : `simpleCPUv1d.m4`. If macros are used you must submit a plain text file : `build.txt`, containing clear instructions of how this file will be used in the build process.

This assembly language file will be assembled and added to the base ISE project from the Exam folder on the VLE for testing. Do **not** submit the full ISE project, or the output image. The encrypted output image will be regenerated during the marking process. Files that do not comply to the stated file formats will be awarded a zero mark.

Q3 (20 marks)

Optimise : rewrite your solution for question 2 using new instructions and hardware to improve your solution's processing performance.

Marks are awarded for:

- Processing performance (15 marks) : how quickly your program performs this task. Marks will be deducted for incorrect or missing RTL descriptions of any new instructions used.
- Minimising program size (5 marks) : minimising storage i.e. instructions and program data. Image data is not included in this calculation.

Submission: for this question you should submit:

- `encrypt.asm` : a plain text file, assembly language program used to implement the encryption functionality.
- You may submit a single macro file : `simpleCPUv1d.m4`. If macros are used you must submit a plain text file : `build.txt`, containing clear instructions of how this file will be used in the build process.
- Any ISE schematics (`.sch` files) that have been modified to implement this system. **Note**, remember the schematic restrictions defined on page 2.
- Any new ISE schematics (`.sch` files) and symbols (`.sym` files) that have been created to implement this system.
- The file `decoder.vhd` if you have implement any new instructions. You must also submit a plain text file : `instructions.txt`, containing the RTL descriptions of the micro-instructions used to implement these new instructions during the Fetch, Decode and Execute phases, and a brief describe of their operation.

These files will be added to the base ISE project from the Exam folder on the VLE for testing. Do **not** submit the full ISE project, or the output image. The output image will be regenerated during the marking process. Files that do not comply to the stated file formats will be awarded a zero mark.

Q4 (20 marks)

Arithmetic function : to process image data a range of arithmetic functions are required. To convert a colour image into a grayscale image i.e. replace each pixel's RGB value with its equivalent brightness (Y), we can use the following equation:

```
Y = (R + G + B) / 3
Y = (R/3) + (G/3) + (B/3)
```

This brightness or luminance calculation requires a divide by 3 function. Write a software subroutine: `divide`, to implement this arithmetic function. The accumulation of the RGB pixels is passed to this function in register RA and the result returned in register RA. The remainder of this divide function is not used.

Note, this solution may be implemented using new instructions and hardware.

This function will be tested using the following test program: `divide.asm`.

```
start:
     move RA 0x7F         # set dummy pixel values
     call divide          # divide 0x7F/3=0x2A

     store RA 0xFFF       # signal subroutine finished

     move RA 0x0F         # set dummy pixel values
     call divide          # divide 0x0F/3=0x05

     store RA 0xFFF       # signal subroutine finished

     move RA XXX          # random value XXX assigned
     call divide          # during testing

     store RA 0xFFF       # signal subroutine finished

trap:
     jumpu trap           # end

divide:
     …                    # add your code here
     ret

data:
     …                    # add your data/variables here
```

Figure 5 : divide test program

Download the `divide.asm` test program from the Exam folder on the VLE, and add your subroutine code and variables as indicated. The first two test values processed are 0x7F and 0x0F. The third value is undefined at this time, but will be a random 8bit value. The result of the division subroutine is stored to address 0xFFF to simplify location within the waveform diagram i.e. the GPIO chip select line will go high.

Marks are awarded for:

- Functionality (5 marks) : correct execution of the test program
- Processing performance (10 marks) : how quickly your program performs this task. Marks will be lost for incorrect or missing RTL descriptions of any new instructions used.
- Minimising program size (5 marks) : minimising storage i.e. instructions and program data. Image data is not included in this calculation.

Submission: for this question you should submit:

- `divide.asm` : a plain text file, assembly language program used to implement the divide functionality. You must use the assembly language template shown in figure 5.
- You may submit a single macro file : `simpleCPUv1d.m4`. If macros are used you must submit a plain text file : `build.txt`, containing clear instructions of how this file will be used in the build process.
- Any ISE schematics (`.sch` files) that have been modified to implement this system. **Note**, remember the schematic restrictions defined on page 2.
- Any new ISE schematics (`.sch` files) and symbols (`.sym` files) that have been created to implement this system.
- The file `decoder.vhd` if you have implement any new instructions. You must also submit a plain text file : `instructions.txt`, containing the RTL descriptions of the micro-instructions used to implement these new instructions during the Fetch, Decode and Execute phases, and a brief describe of their operation.

These files will be added to the base ISE project from the Exam folder on the VLE for testing. Do **<u>not</u>** submit the full ISE project. The output image will be regenerated during the marking process. Files that do not comply to the stated file formats will be awarded a zero mark.

Appendix A : SimpleCPUv1d instruction-set

```
# INSTR   IR15 IR14 IR13 IR12 IR11 IR10 IR09 IR08 IR07 IR06 IR05 IR04 IR03 IR02 IR01 IR00

# MOVE    0    0    0    0    RD   RD   X    X    K    K    K    K    K    K    K    K
# ADD     0    0    0    1    RD   RD   X    X    K    K    K    K    K    K    K    K
# SUB     0    0    1    0    RD   RD   X    X    K    K    K    K    K    K    K    K
# AND     0    0    1    1    RD   RD   X    X    K    K    K    K    K    K    K    K

# LOAD    0    1    0    0    A    A    A    A    A    A    A    A    A    A    A    A
# STORE   0    1    0    1    A    A    A    A    A    A    A    A    A    A    A    A
# ADDM    0    1    1    0    A    A    A    A    A    A    A    A    A    A    A    A
# SUBM    0    1    1    1    A    A    A    A    A    A    A    A    A    A    A    A

# JUMPU   1    0    0    0    A    A    A    A    A    A    A    A    A    A    A    A
# JUMPZ   1    0    0    1    A    A    A    A    A    A    A    A    A    A    A    A
# JUMPNZ  1    0    1    0    A    A    A    A    A    A    A    A    A    A    A    A
# JUMPC   1    0    1    1    A    A    A    A    A    A    A    A    A    A    A    A

# CALL    1    1    0    0    A    A    A    A    A    A    A    A    A    A    A    A

# OR      1    1    0    1    RD   RD   X    X    K    K    K    K    K    K    K    K  -- NOT IMPLEMENTED
# XOP1    1    1    1    0    U    U    U    U    U    U    U    U    U    U    U    U  -- NOT IMPLEMENTED

# RET     1    1    1    1    X    X    X    X    X    X    X    X    0    0    0    0
# MOVE    1    1    1    1    RD   RD   RS   RS   X    X    X    X    0    0    0    1
# LOAD    1    1    1    1    RD   RD   RS   RS   X    X    X    X    0    0    1    0  -- REG INDIRECT
# STORE   1    1    1    1    RD   RD   RS   RS   X    X    X    X    0    0    1    1  -- REG INDIRECT
# ROL     1    1    1    1    RSD  RSD  X    X    X    X    X    X    0    1    0    0

# ROR     1    1    1    1    RSD  RSD  X    X    X    X    X    X    0    1    0    1  -- NOT IMPLEMENTED
# ADD     1    1    1    1    RD   RD   RS   RS   X    X    X    X    0    1    1    0  -- NOT IMPLEMENTED
# SUB     1    1    1    1    RD   RD   RS   RS   X    X    X    X    0    1    1    1  -- NOT IMPLEMENTED
# AND     1    1    1    1    RD   RD   RS   RS   X    X    X    X    1    0    0    0  -- NOT IMPLEMENTED
# OR      1    1    1    1    RD   RD   RS   RS   X    X    X    X    1    0    0    1  -- NOT IMPLEMENTED
# XOR     1    1    1    1    RD   RD   RS   RS   X    X    X    X    1    0    1    0
# ASL     1    1    1    1    RD   RD   RS   RS   X    X    X    X    1    0    1    1  -- NOT IMPLEMENTED

# XOP2    1    1    1    1    RD   RD   RS   RS   X    X    X    X    1    1    0    0  -- NOT IMPLEMENTED
                                                                                       -- REG INDIRECT

# XOP3    1    1    1    1    RD   RD   RS   RS   X    X    X    X    1    1    0    1  -- NOT IMPLEMENTED

# XOP4    1    1    1    1    RD   RD   RS   RS   X    X    X    X    1    1    1    0  -- NOT IMPLEMENTED
                                                                                       -- REG INDIRECT

# XOP5    1    1    1    1    RD   RD   RS   RS   X    X    X    X    1    1    1    1  -- NOT IMPLEMENTED
```

0 = Logic 0, 1 = Logic 1, U = Undefined, X = Unused, A = Address, K = Constant.

RD = Destination register, RS = Source register.

Functionality of XOP1, XOP2, XOP3, XOP4 and XOP5 are not defined.