School of Built Environment, Engineering and Computing

Module Handbook

Module name and CRN Software and Systems CRN 11550

Module Leader
Mario Donato Marino
Semester
B
Level
7

ASSIGNMENT TITLE:	ARM Processor disassembler and emulator
ASSIGNMENT WEIGHTING:	100%
HAND-OUT DATE:	Week 1 (w/c 31st Jan 2022).
SUGGESTED STUDENT EFFORT:	40 hours approx.
SUBMISSION DATE:	Sunday 15th May 2022 at 23:59.

SUBMISSION INSTRUCTIONS:

Submission of program code should be via the VLE, followed by demonstration to the tutor within the subsequent lab session (17th May 2022). Your program based solutions should be zipped up into a single file in the form `<studentID>.zip`. You MUST demonstrate your work in order to receive a mark.

FEEDBACK MECHANISM:

You will be provided with verbal feedback regarding your work during the lab based demonstrations.

LEARNING OUTCOMES ADDRESSED BY THIS ASSIGNMENT:

1. Be able to apply and critically evaluate programming approaches which are appropriate to the development of systems level software.

2. Develop a working knowledge of the relationship between software and the underlying hardware systems.

3. Have an ability to identify, select and use tools and associated tool-chains which are appropriate for the development of low level and embedded type systems.

NOTES:

The usual University penalties apply for late submission. This is an individual assessment. Submission of an assessment indicates that you, as a student, have completed the assessment yourself and the work of others has been fully acknowledged and referenced.

By submitting this assessed work, you are declaring that you are fit to submit, and you will therefore not normally be eligible to submit a request for mitigation for this work.

If your result for this assessment is recorded as Non-Submission or your mark for this assessment and for the whole module is below 40%, you will have opportunity to take reassessment with a submission date on July 2022 to be announced on the VLE (see Reassessment information below). If you are granted deferral through the mitigation process, you may complete the reassessment with a full range of marks available.

If you fail to attend the demonstration at the scheduled date and time without agreed mitigation, you will be given one further opportunity to demonstrate your work (incurring a 5% late penalty) at a time scheduled by the module team. If you miss this second opportunity, your result will be recorded as Non-Submission.

For further information, please refer to your Course Handbook or University Assessment Regulations.

DETAILS OF THE ASSESSMENT

In order to receive a mark for your work you are required to do a practical demonstration during the timetabled sessions, or by special arrangements with the module tutors (in exceptional circumstances only). Handing in the source code only will not provide you with a mark. During the demonstration you will be asked to explain certain aspects of your program code. If you are unable to explain this to a satisfactory level then you will not be given credit for the work. i.e. the mark is awarded for the demonstration and your ability to discuss the solution, not for the submitted code. This also helps address learning outcome 1 of the module.

Assignment Overview

The overall aim of your assignment is to build a software based implementation of a Central Processing Unit (CPU). Such an implementation should be capable of emulating the execution of machine code instructions. The implementation should be written in the ‘C’ programming language preferably but you are free to use ‘C++’, Java, Python or any other language (if you are more comfortable with the latter ones). You are free to use any compiler you wish, however the GNU tool-chain is recommended. CPUs can clearly be very complex devices, hence you will not be expected to build a fully compliant implementation. However the more complete the implementation, the higher the final grade will be. The CPU to be simulated will be loosely based on the standard ARM

architecture. You will be provided with a datasheet which will specify the nature of the device (e.g. register count, status register format etc. and the instruction set to be supported). In order for the simulation to work your program will also have to provide some virtual peripheral devices. e.g. Some virtual RAM will be required and virtual display?.

The project has been broken down into three requirements described as follows.

Requirement 1 – Instruction Decoding and Disassembling: 40 Marks

The first requirement will be to develop a program which is capable of decoding ARM based instructions from their native form (machine code) and displaying these as Assembly language instructions. This should result in the development of a computer program which has the ability to fetch, decode and display the instructions within the provided instruction set.

One key aspect you will have to consider is how the instructions will be defined and read by your program. The op-codes should be stored in your virtual RAM (or ROM if you decide to include such a feature). However these codes need to get into the virtual machine somehow. Initially you may want to consider hard-coding some simple instructions for testing purposes. Later you may decide to load the instructions from a file.

As the program is executing it should display the original op-code value along with the assembly language equivalent. For example, given the following machine code hexadecimal op-codes -

```
E3 A0 00 01
E3 A0 10 02
E0 80 20 01
E2 82 20 05
```

The program should output something similar to the following.

```
Op-Code          Assembly Mnemonic
-----------------------------------
E3 A0 00 01      MOV r0,#1
E3 A0 10 02      MOV r1,#2
E0 80 20 01      ADD r2,r0,r1
E2 82 20 05      ADD r2,r2,#5
```

Requirement 2 – Basic Emulation: 30 Marks

The second requirement of the assignment is to extend the implementation so that as well as displaying each Assembly language instruction it should also emulate the behaviour of these. i.e. implement a basic CPU emulator. For this requirement only the basic ALU type instructions need to be supported, e.g. ADD, SUB, CMP, MOV. Also conditional execution of each instruction does NOT yet need to be supported.

In order to achieve this you will need to create variables to represent the internal “state” of

the CPU. For example, you will need to be able to store the current contents of each of the 16 registers. It is recommended that all registers are initially set to 0 (except R13 which should be set to the top of stack position).

As the program is executing it should display the same output as requirement 1, plus the contents of each of the sixteen registers. This way the correct execution of each instruction will be verifiable. It may be a good idea to pause between each instruction and wait until the user presses the enter key before proceeding. For example -

```
Op-Code          Assembly Mnemonic
-----------------------------------
E3 A0 00 01      MOV r0,#1

R0: 00000001     R1: 00000000  r2: 00000000 r3: 00000000
R4: 00000000     R5: 00000000  r6: 00000000 r7: 00000000
R8: 00000000     R9: 00000000  r10:00000000 r11:00000000
R12:00000000     R13:00010000  r14:00000000 r15:00000104

Press <return> for next instruction.

Op-Code          Assembly Mnemonic
-----------------------------------
E3 A0 10 02      MOV r1,#2


R0: 00000001     R1: 00000002  r2: 00000000 r3: 00000000
R4: 00000000     R5: 00000000  r6: 00000000 r7: 00000000
R8: 00000000     R9: 00000000  r10:00000000 r11:00000000
R12:00000000     R13:00010000  r14:00000000 r15:00000108

Press <return> for next instruction.

Op-Code          Assembly Mnemonic
-----------------------------------
E0 80 20 01      ADD r2,r0,r1

R0: 00000001     R1: 00000002  r2: 00000003 r3: 00000000
R4: 00000000     R5: 00000000  r6: 00000000 r7: 00000000
R8: 00000000     R9: 00000000  r10:00000000 r11:00000000
R12:00000000     R13:00010000  r14:00000000 r15:0000010C

Press <return> for next instruction.

Op-Code          Assembly Mnemonic
-----------------------------------
E2 82 20 05      ADD r2,r2,#5

R0: 00000001     R1: 00000002  r2: 00000008 r3: 00000000
R4: 00000000     R5: 00000000  r6: 00000000 r7: 00000000
R8: 00000000     R9: 00000000  r10:00000000 r11:00000000
R12:00000000     R13:00010000  r14:00000000 r15:00000110

Press <return> for next instruction.

Program end. Terminating.
```

Requirement 3 – Full Emulation: 30 Marks

The final requirement will be to extend requirement two so that it emulates all of the provided instruction set. So therefore the branching instructions (B and BL) will need supporting as will the memory access instructions (LDR and STR), along with all instructions defined within the datasheet.

All instructions should also be extended to support conditional execution (based on condition codes). Therefore you should take care to ensure that the instructions not only undertake their operation correctly, but they also carry out appropriate side effects, such as the setting of status register flags.

To get to the highest bracket of the available marks you should also include the ability to count elapsed time during the simulation (in terms of instruction cycle usage). To implement this feature you will need to use the instruction timing information provided in the datasheet and keep a running total of total time elapsed.

MARKING SCHEME / CRITERIA

The following criteria will be taken into account during the demonstration and grading of your assignment solution. An inability to explain the code can result in a fail even if requirements are completed.

distinction (>70%)	Merit (60%+)	Pass (40%)	Fail <40%
Requirements 1 and 2 fully completed along with a very good attempt of requirement 3.	Requirements 1 and 2 completed with only minor issues (if any).	Requirement 1 and 2 completed to a usable level, but lacking some functionality.	Requirements 1 and/or 2 are not completed to a satisfactory level.
Code is well formatted and commented.	Code is generally well formatted and commented.	Code is generally understandable, but formatting could be better.	Code does not compile.
Output is very well presented.	Output is well presented.	Output is adequate.	No emulation output.
Explanations to all questions are clear and concise.	Explanations to most questions are clear and concise.	Explanations to most questions are clear and concise.	Explanations to some questions are incorrect or reflect lack of knowledge regarding the solution.

EXAMPLE OF SIMULATOR
https://schweigi.github.io/assembler-simulator/

LATE SUBMISSION OPPORTUNITIES

If you fail to submit the electronic copy of your work via the VLE by the specified date, then the usual University penalties for late submission will apply.

If you fail to demonstrate your work in the lab session following submission then you will be given one more chance to demonstrate (incurring a 5% late penalty). If you fail to demonstrate on your second opportunity however then you will have a non-submission recorded for this assignment.

REASSESSMENT and DEFERRAL OPPORTUNITIES

Reassessment and deferral will take the form of resubmission, based on the original assignment specifications. The practical demonstrations will take place following submission. If you are granted deferral through the mitigation process, you may complete the reassessment July 2022 (announced on the VLE) with a full range of marks available.