UG Exam Performance Feedback
First Year
2017/2018 Semester 1

COMP15111 Fundamentals of Computer Architecture Jim Miles

Comments

Overall the exam was well answered and the average mark was high. Considering that this was the first year of a new format and that the compulsory questions spanned the whole syllabus it is clear that most students had worked hard and had prepared well for the exam.

Q1. Methods and the stack
This question was generally answered well, with high average marks. There were a few errors that were made by several students:

a)Where decimal was converted to binary some students read the binary number in the wrong direction and therefore got the wrong answer, some students omitted the final division (1/2 = 0r1).
b)Some students omitted either the octal or the hexadecimal values.

Q2. Methods and the stack

a)A significant number of students interpreted the question as being about performance and speed. Performance and speed can be reasons for choosing a load-store architecture over a different architecture but the question does not ask for a comparison of architectures. In a load-Store architecture load and store operations are separated from data processing operations and the data processing operations must have access to the data without loading or storing it within the instruction. For that reason there must be registers in the processor.

b)This was generally well answered however a significant number only gave the operation (e.g. LDR), the instruction needs both the operation and the operand, in this case a label to specify where to load or store from.

c)(i) This was generally well answered with a minority calculating the wrong number of bits required to identify one of 16 registers.
(ii) A variety of answers were provided. Most identified the fact that if more than 16 registers were used an extra bit, 5 or more bits per register would be needed for 3-register operations. The answer was expected to include the 32-bit length of the opcode and to discuss the way that increasing the number of bits required to specify the registers would decrease the number of bits available to specify the operation, including conditional execution, setting the flags or not, or space for immediate values. The fixed 32-bit instruction length means that there must be a trade-off between the number of registers and number of instructions in the instruction set.

Q3. Code and data structures

Q3 was generally answered well with a high average mark. There were some points on which a number of students lost marks:

a) A number of students mixed up EQU and DEFW, specifying that EQU is an instruction for variable declaration and DEFW is to create a constant. A number also gave the MOV instruction for creating either variables or constants.

b)When reading a string, a lot of students loaded the string value directly to the register rather than loading the address of the string. The ADRL instruction was frequently misspelled. In the example code there was confusion with the sequence of the instructions (LDR, CMP, SVC), and correct usage of the addressing mode, as a result some programs would print out first character twice, or skip the first character, print only first character etc.

c)(i) Some students wrote “indirect addressing”, without specifying a mode.
(ii) Quite a few answers did not explain why the addressing mode is particularly suitable for arrays.

Q4. Methods and the stack

Generally this question was answered well showing good knowledge and understanding.

a)The most common reasons for losing marks were diagrams that did not show either example addresses or the direction in which the address gets larger, diagrams that did not show the effect of push and pop on the SP and lack of clarity that the stack grows towards lower addresses.

b)Was generally very well answered, particularly b (ii) where most students obtained full marks.

c)Whilst most students secured some marks here there were a number of answers that presented either misconceptions or irrelevant information. Many students expressed the view that the stack is in some manner safer than registers, that registers might be accidentally or unknowingly overwritten. While it is true that register contents must be carefully managed, registers should not be overwritten by code unless it is known that it is safe to do so.

Documentation of methods should always make clear which registers are affected by methods or code blocks and in what manner, in order to ensure that information is not overwritten unknowingly. The stack and registers can both be overwritten in a single instruction if the program is wrongly coded and so the stack is not really any more secure or safe than registers.

Similarly the complexity of tracking the position of information in the stack or of ensuring that information in registers is not lost is not a significant disadvantage provided that code is carefully designed and properly documented. Access to registers is much faster than access to memory and so passing parameters in registers is faster than passing parameters by the stack for which a store and a load would be required for each parameter passed into a method. Speed is an advantage for registers and a disadvantage for the stack.

The number of general purpose registers available for data is limited (R0 – R12) and some of these will be needed for intermediate results etc within the method so that the number of parameters that can be passed in registers is small. Any method that has many parameters cannot use registers to pass them. This is a disadvantage for registers and an advantage for the stack.

Other valid advantages/disadvantages were accepted.

Q5. The operating system and accessing peripherals

This question was answered reasonably well but the average mark was lower than the rest of the questions. This was the last question on the paper and the impression gained during marking was that many students either ran out of time during this question or completed it in haste.

a)Generally answered well. Polling and interrupts were the expected answers, some students gave DMA as an answer which was equally accepted and marked accordingly. A number of answers included statements that polling is slow which is not necessarily true as the processor could keep polling the status register in a tight loop and the response time could be faster than interrupts. Polling does slow down any other processes that are running but it is not slow in itself.

b)Most answers gave an overview but many omitted some or all details of how the ISR would achieve requirements such as saving and restoring the state, and returning to the correct address in the interrupted code.

c)There were a rather small number of good answers to the final part of the final question, this may have been a time issue.