Digital Systems: Revision questions

Mike Spivey, Trinity Term, 2022

Assembly language: addressing

1 [resit19/1] This question asks for subroutines written in assembly language for a typical RISC machine; full marks will be given for answers that show in general terms how the desired functions may be implemented, even if they do not strictly conform to the conventions of any particular machine. You should explain in words the meaning of any instruction that might otherwise be unclear.

(a) Write a subroutine that takes a single 32-bit argument, and uses a loop containing shifts and bitwise logical instructions to compute its *popcount*, the number of one bits it contains. Thus, for example, the value `0x11223344` has a popcount of 10. [5 marks]

Figure 1 shows the beginning and end of a table of 256 bytes, giving the number of one bits in each possible single-byte value.

(b) Write a subroutine containing no loops that uses this table four times to compute the popcount of each byte of its 32-bit argument, and returns the sum of these values as the popcount of the whole argument. Briefly explain the addressing modes that appear in the instructions you use to access the table. [5 marks]

(c) Assess the relative running times of these two implementations of *popcount* in the worst case, stating any assumptions you make. [4 marks]

(d) Describe a different table of one-byte values (you need not write out all of it), and use it to write a subroutine *countlz* that quickly finds the number of leading zero bits in its argument. For example, with an argument of `0x01234567`, the subroutine should return 7. [6 marks]

2 [part of 2008/3] This question concerns assembly-language programming in Thumb code. *You need not be concerned with the details of instruction mnemonics, provided the meaning of each instruction is clear and they fit into the overall architecture of the machine.*


```
    .align 2
popbyte:
    .byte 0, 1, 1, 2, 1, 2, 2, 3, 1, 2, 2, 3, 2, 3, 3, 4
    .byte 1, 2, 2, 3, 2, 3, 3, 4, 2, 3, 3, 4, 3, 4, 4, 5
    .byte 1, 2, 2, 3, 2, 3, 3, 4, 2, 3, 3, 4, 3, 4, 4, 5
    .byte 2, 3, 3, 4, 3, 4, 4, 5, 3, 4, 4, 5, 4, 5, 5, 6
    @ ... 10 lines omitted ...
    .byte 3, 4, 4, 5, 4, 5, 5, 6, 4, 5, 5, 6, 5, 6, 6, 7
    .byte 4, 5, 5, 6, 5, 6, 6, 7, 5, 6, 6, 7, 6, 7, 7, 8
```

Figure 1: *Population count table for Question 1*

The following program fragment determines the maximum value in an array $a[0\,..\,N)$ of positive integers; it returns 0 if $N = 0$.

```
int i, m;
static int a[N];

i = 0; m = 0;
while (i < N) {
    if (a[i] > m) m = a[i];
    i = i+1;
}
```

A partial translation of this program fragment into Thumb code is as follows:

```
    mov r0, #0          @ i = 0
    mov r1, #0          @ m = 0
    b test
loop:
    ...                 @ lines not shown
test:
    cmp r0, #N
    blt loop
```

(a) Complete the translation of the program fragment into assembly code, keeping the variables i and m in registers. Assume that the base address A of array a occupies 32 bits, while the constant N fits into 8 bits. [8 marks]

(b) Show how the program can be speeded up by keeping the address of $a[i]$ in a register. [4 marks]

3 [2020/2] Three hardware engineers, Ali, Beth and Chad, have built an interface to a remote sensor that unfortunately delivers its results with all the bits reversed, so that a reading that should be `0x12345678` (or

0001 0010 0011 0100 0101 0110 0111 1000

in binary) in fact appears as `0x1e6a2c48` or

0001 1110 0110 1010 0010 1100 0100 1000.

They have decided to correct the error by implementing a subroutine with the C heading

```
unsigned revbits(unsigned x)
```

that reverses the bits in its argument *x*. To be sure of the best performance, they will write the body of the subroutine in assembly language for a simple RISC-based microcontroller.

Each engineer suggests a different approach: fill in enough of the details of each to evaluate its efficiency in terms of execution time and size in ROM. *You need not write out all the assembly-language code for each approach, provided you give enough detail to support your estimate of its performance.*

(a) Ali suggests the simple approach of taking the bits of *x* one at a time and appending them to a growing answer in the correct order. [6 marks]

(b) Beth's idea is that it will be faster to make a table in ROM that contains the reversal of individual bytes, and use this four times to reverse the bytes in *x*, combining the results by shifting and or'ing. [6 marks]

(c) Chad observes that alternating blocks of four bits in a word can be picked out using the masks `0x0f0f0f0f` and `0xf0f0f0f0`, and by shifting and combining the results, each block can be swapped with its neighbour. Doing this successively for blocks of 1, 2, 4, 8 and 16 bits will result in reversing all the bits. He points out that some versions of the microcontroller have an instruction

```
rev ra, rb
```

that sets register `ra` to the result of reversing the value in `rb` byte-wise, so that `0x12345678` is mapped to `0x78563412`, and this does part of the job in one instruction. [6 marks]

(d) Briefly compare the strengths of each approach. [2 marks]

4 Another sensor that produces 16-bit readings suffers from the same problem as in the previous question. An array of readings from that sensor is stored in an array, and we need to reverse each reading in a similar way. Assuming the subroutine `revbits` implemented in previous parts of the question, write (first in C then in assembly language) a subroutine `revarray` with heading

```
void revarray(unsigned short arr[], int n)
```

that reverses each of the `n` elements of the array whose base address is passed as `arr`.

Computer arithmetic

5 [2010/4] A certain microprocessor has a 32-bit datapath and uses both unsigned binary and twos-complement signed representations of numbers. There are instructions `add` and `sub` that add and subtract the contents of two registers, storing the results in a third register. There also a comparison instruction that subtracts two registers and sets the four flags NZCV from the result, and instructions `blt` and `blo` that branch if the flags indicate that the first register is less than the second in value. The `blt` instruction interprets its inputs as twos-complement signed numbers, and `blo` interprets them as unsigned numbers.

(a) Define two functions that map sequences of 32 bits to the integers they represent in the twos-complement and in the unsigned representation. [2 marks]

(b) Specify precisely the function of an adder that adds two 32-bit unsigned numbers to obtain a 32-bit result, and show that the same adder can be used with twos-complement numbers. [4 marks]

(c) Show how to combine a 32-bit adder with appropriate additional logic to make a circuit that performs 32-bit subtraction. [4 marks]

(d) Give an example to show that the `blt` and `blo` instructions can give different results. [4 marks]

(e) Give an example to show that the `blt` instruction cannot simply use the sign bit that results from a 32-bit subtraction. Describe appropriate logic to compute the result, and show that it is correct. [4 marks]

(f) Design logic to give the correct result for the `blo` instruction, and show that it is correct. [2 marks]

6 [2019/1]

(a) Define the function $bin(a)$ that maps a bit-vector a of length n to the integer it represents in the unsigned representation. [2 marks]

(b) Specify the meaning of the carry flag that is output by an add instruction (written `adds` in Thumb code) by writing an equation involving *bin* that relates the inputs x and y to the output z and carry output C. Repeat this for the subtract instruction `subs`. [2 marks]

(c) In addition to the normal add instruction, many machines also have an add-with-carry instruction (written `adcs`) that adds together the contents of two registers and also adds on the value of the carry flag. A 64-bit integer can be stored in two 32-bit registers by putting the 32 high-order bits in one register and the 32 low-order bits in the other. If two 64-bit integers are stored in this way in four registers, show how the `adds` and `adcs` instructions can be combined to add one 64-bit integer to the other, and prove that this code computes the correct result. [4 marks]

(d) Describe the operation performed by an `sbcs` (subtract-with-carry) instruction that can be combined with `subs` to perform a 64-bit subtraction, and explain how the `sbcs` instruction can be implemented using a 32-bit adder. [4 marks]

(e) Figure 2 shows an algorithm that performs 32-bit unsigned division, assuming $b > 0$, and returning a result q such that $0 \le a - q \times b < b$. It maintains the invariant

$$a \times 2^{32} = q \times b \times 2^{j+32} + r \times 2^j,$$

with $0 \le r \times 2^j < b \times 2^{32}$. The variable r is stored in 64 bits and all the others in 32 bits. Implement this algorithm in assembly language for a typical RISC machine with 32-bit registers that provides the `adcs` and `sbcs` instructions among others. Show the correspondence between instructions in your program and operations in the algorithm. If you like,

```
unsigned udiv(unsigned a, unsigned b) {
    unsigned q = 0;
    unsigned64 r = a;
    unsigned j = 32;

    do {
        r = r + r; q = q + q;
        if (r >= b * 2^32) {
            q = q + 1;
            r = r - b * 2^32;
        }
        j = j - 1;
    } while (j > 0);

    return q;
}
```

Figure 2: *Algorithm for question 6*

you can reuse the register in which a arrives as one of the two registers used to store r. For full marks, minimise the number of registers used, the number of instructions in the loop, and the number of taken branches per iteration. [8 marks]

Devices and interrupts

7 Figure 3 shows the wiring of a common-cathode LED digit display with seven segments, and Figure 4 shows four such digits connected to a microcontroller. There is an n-type MOSFET controlling the cathode of each display, so that a high level on a certain GPIO pin permits current to flow through the digit. The anodes of corresponding segments in each digit are connected together and controlled by seven more GPIO pins, making 11 pins in all. These GPIO pins correspond to the 11 least significant bits of the device register `GPIO_OUT`. In order to display an four-digit number, the digits are illuminated one at a time in rapid succession.

(a) The circuit in Figure 4 is missing certain components needed for it to function properly. Identify these components and where they would be placed in the circuit. Suggest approximate values for the components. [4 marks]

(b) The display is intended for a debugging output from an application, able to display any 16-bit value in hexadecimal. There will be a subroutine

```
void set_display(unsigned val);
```

that sets the value shown in the display. Additionally, the microcontroller has a timer that causes a regular interrupt, and the

```
void next_digit(void);
```

on each interrupt. Design and implement these two subroutines. [8 marks]

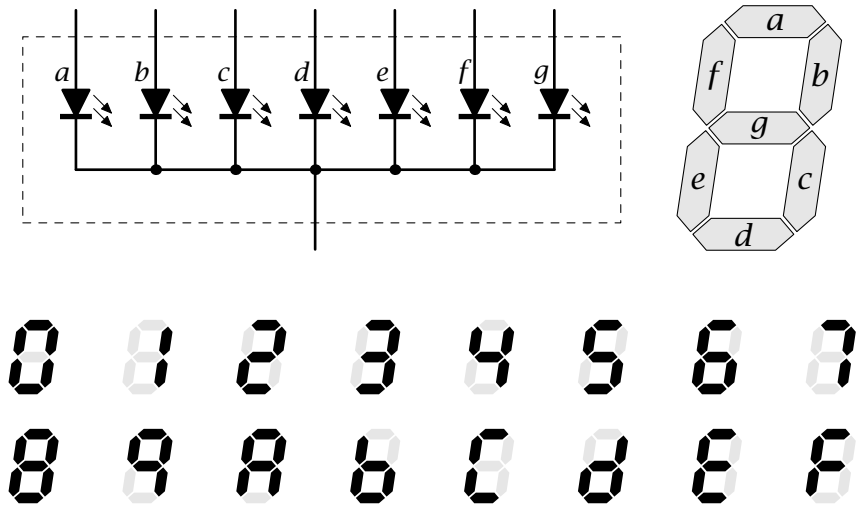

Figure 3: *Seven-segment display*

(c) Smooth interaction between the program and the interrupt handler may depend on marking some program elements as `volatile` or disabling interrupts at some point. Discuss whether these measures are needed in your implementation of the display software, and what might be the consequence if they were omitted. What might be the effect on the display if interrupts were disabled for a significant time in other parts of the program? [3 marks]

(d) Suggest a suitable interval between timer interrupts for an acceptable display, and estimate roughly what proportion of the CPU time (for a microcontroller like the micro:bit) would be spent on updating the display. [3 marks]

8 A security device for colleges is designed to turn on an indicator light in the porter's lodge for one second each time a person is detected coming through the college gate. If two or more successive people come through the gate, the light emits a one-second flash for each of them, with a gap of one second between each flash and the next.

The hardware design of the device is as follows. The device has a microcontroller with an I/O register `EYE_DETECTED` that is set to 1 when a person is detected, causing an interrupt that calls a handler named `eye_handler`. The interrupt handler must set `EYE_DETECTED` back to 0 before another person can be detected. Another I/O register `LIGHT` can be set to 1 to put the light on, and 0 to put it off again.

(a) Discuss the design of the software for the microcontroller, showing the main features of the code for the interrupt handler and the main program. Evaluate also the options for implementing the one-second delays that are needed to control the light. [10 marks]

(b) The device sells in sufficient numbers that the manufacturer decides to put out an improved version that has several detectors

`EYE_DETECTED[0]`, `EYE_DETECTED[1]`, ...,

all sharing the same interrupt, and several lights

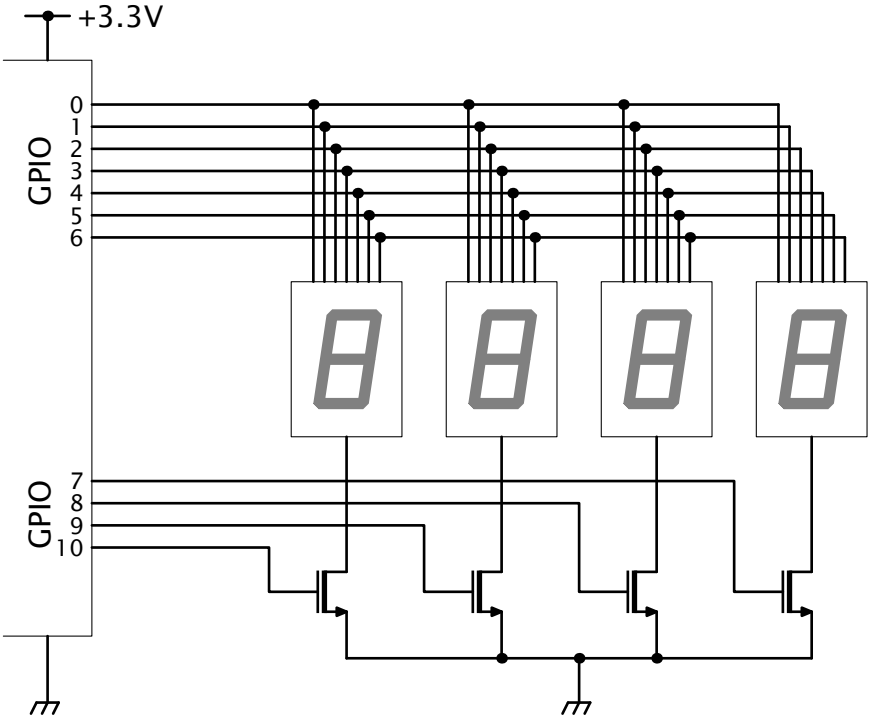

Figure 4: *Multiple digits with a microcontroller*

`LIGHT[0]`, `LIGHT[1]`, ...

that should each flash for one second when the corresponding detector is activated. Describe how the software design could be enhanced for the new device, and outline the code needed to implement the changes. [10 marks]

9 Some 'friends' of yours are starting an indoor gardening project, and you have agreed to build a soil temperature monitor for them in return for a share of the produce. The hardware design is complete, with temperature sensors connected to an analogue-to-digital converter (ADC) in your chosen microcontroller. After configuration, the ADC is started by setting a device register `ADC_START` to 1. Some time later, the ADC sets event register `ADC_DONE` to 1 and causes an interrupt that calls the handler function `adc_handler`. The reading is then available in device register `ADC_DATA`. After `ADC_DONE` is cleared, the device will not request another interrupt before it is started again. The ADC has eight channels, selected by assigning an integer from 0 to 7 to the device register `ADC_CHANNEL`; this must done before starting the ADC.

In the existing design, there is one temperature sensor connected to channel 0 of the ADC, and the control software is single-threaded. Design an interrupt-controlled driver for the ADC, providing a function

```
int adc_reading(void)
```

that takes a reading and returns its value.

Embedded operating system

10 Following the success of the first growing season, your friends from question 9 decide to expand the project, and now need a temperature monitor that supports multiple sensors. You decide to connect the sensors to different channels of the same ADC, and to use a message-based process scheduler to organise the software, including multiple processes that each monitor the temperature of a different sensor. Design a driver process for the ADC, providing a function

```
int adc_reading(int channel)
```

the may safely be called by other processes to read the temperature on a specified channel, ensuring that one reading is finished before another one starts.

11 [resit19/2] The Nordic processor die has a temperature sensor that can be activated when necessary by setting `TEMP_START = 1`, and interrupts some time later on `TEMP_IRQ` with `TEMP_DATARDY = 1`; the temperature reading (in units of 1/4 Celsius) can then be fetched from the register `TEMP_TEMP`.

Design a driver process for the sensor, suitable for use with up to five client processes, which perhaps may wish to search for primes less assiduously or flash the beating heart less passionately, if the processor is getting too hot for comfort. When no client process is requesting the temperature, the sensor should remain idle. As soon as a request comes, the sensor should be started, and any other requests that come in subsequently should receive the same reading when it is ready. Later requests should result in a fresh reading.

If the same reading is being sent to several client processes, and one of them should have exited by the time the reading is ready, discuss the consequences for the future functioning of the program.

12 The NRF51822 has a hardware random number generator. When appropriately configured, it periodically generates an interrupt with handler `rng_handler`, and an eight-bit random number can then be retrieved from the device register `RNG_VALUE` before resetting the event flag `RNG_VALRDY` to zero. In micro:bian, a driver process can call `connect(RNG_IRQ)` to connect to the interrupt.

(a) Design a micro:bian driver process for the random number generator, and write a subroutine

```
unsigned randbyte(void);
```

that can be called from client processes in order to fetch a random byte. As in the previous exercise, use a stack to hold a limited number of random values that have been generated but not used.

(b) Experience with the program indicates that clients are likely to call `randbyte()` several times to fetch multiple random bytes. Explain why doing so may cause an unacceptably high overhead, and show how to modify the program in order to provide a subroutine

```
unsigned randint(void);
```

that returns a random four-byte integer more efficiently than using four calls to `randbyte`.

13 A concurrent program running under micro:bian consists of three processes, *A*, *B*, and *C*.

- Process *A* inputs readings from a sensor using interrupt-based control; it accepts messages of type REQUEST and responds by taking a reading and sending it to the requesting process in a message of type REPLY.
- Process *B* follows the same protocol, accepting requests and sending replies. On receiving each request, it gathers five values from process *A* and replies to its client with the sum of the readings.
- Process *C* repeatedly requests a value from process *B* and prints it, using a device driver for the serial port with a large buffer.

(a) Write code for the body of process *B*. It should respond to requests from any client, but knows the identity of process *A* as a global constant.

(b) If process *A* produces readings at a far slower rate than process *C* can print them, describe the steady state of the system and the sequence of messages sent when process *A* receives an interrupt.

(c) How would it affect the function and efficiency of the system if, instead of waiting for a request before gathering its five readings from *A*, process *B* instead began immediately to gather readings, and accepted a request from a client only when the readings were complete?

Combinatorial and sequential logic

14 [part of resit19/3] Design a fully synchronous sequential circuit that monitors a single input *a* and has a single output *z*. If *a* has been high at two or more successive clock edges and is low at the next clock edge, then the circuit should immediately produce an output pulse for one clock cycle, and it should repeat this behaviour indefinitely. [9 marks]

15 [problem 5.7 extended] A *popcount circuit* has n Boolean inputs, and computes a binary number (with $\lfloor \log n \rfloor + 1$ bits) that counts the number of 1 bits among the inputs.

(a) Show how to construct a popcount circuit from a balanced tree of adders so that the combinational path from each input bit passes through $O(\log n)$ adders before reaching the output.

(b) If we use ripple-carry adders to implement the circuit, a k-bit adder has both size and worst-case delay that are linear in k. Use these facts to estimate the size and propagation delay of the popcount circuit.

(c) In fact, some of the delays in ripple-carry adders are smaller than the estimate $O(k)$, because for $i \le j$, the combinational path from the i'th pair of inputs to the j'th output has length proportional to $j - i + 1$. Use this fact to refine your estimate of the delay of the popcount circuit.

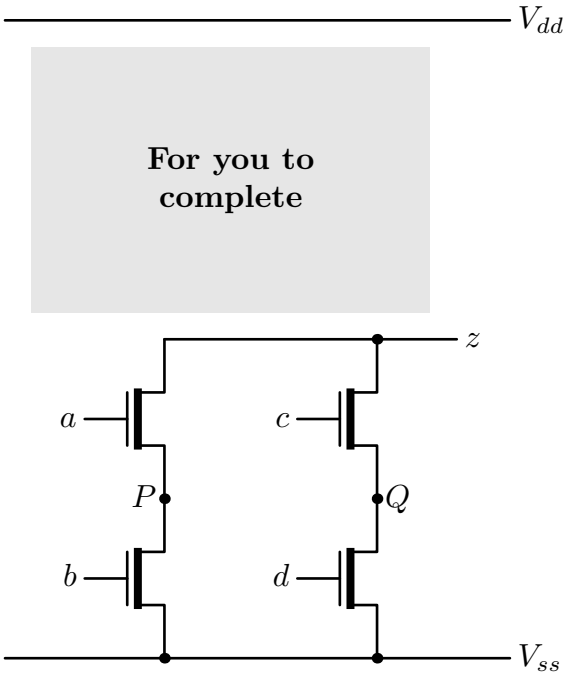

Figure 5: *Circuit diagram for Question 17*

An *n-way priority arbiter* has n input ports $a_0, \ldots, a_{n-1}$, each k bits wide (for fixed k). The output port, also k bits wide, carries the leftmost value a_j such the $a_j \neq 0$, or 0 if all a_j are zero. It could be used in a situation where the n ports carry potential requests for a service, and we wish to select the request that comes in on the lowest-numbered port.

(d) Design an n-way priority arbiter that computes its output with $O(\log n)$ combinational delay. What is the size of the circuit?

16 [2018/1, modified; used as a lecture example in 2021–2] Design a Bit Sequence Recogniser (BSR) circuit that recognises the sequence 1101. The BSR circuit has one data input, one signal output, and one clock input line. When the given sequence has arrived on the serial data input line, the output becomes one for the next clock cycle, otherwise it is zero. The recognised sequences may overlap. You may assume that the serial input signal is synchronised with the same clock signal as the circuit. The example shows the recognition of two overlapping occurrences of 1101.

Data in	0	0	1	1	0	1	1	0	1	0	1	0
Signal out	0	0	0	0	0	0	1	0	0	1	0	0

Use as many state bits as convenient.

17 [part of resit19/3] Figure 5 shows part of the circuit diagram for a CMOS gate with complementary pull-up and pull-down networks that computes output z from inputs a, b, c, d.

(a) Complete the circuit diagram. [3 marks]

(b) Determine the Boolean function that is computed by the gate. [3 marks]

(c) Another CMOS gate has the same pull-down network as in Figure 5, except that the points labelled P and Q are connected to each other. What corresponding change is needed in the rest of the circuit? [2 marks]

(d) Determine the Boolean function that is computed by the second circuit, and find values for the inputs that lead the two circuits to produce different outputs. [3 marks]

Datapath architecture

18 [2019/3] The datapath of a simple microprocessor design contains an ALU and a barrel shifter that both receive inputs from the 32-bit register file. In the existing design, both feed into a multiplexer that allows the output of either of them to be the result of each instruction. The ALU output can also be used as an address to access data memory. There is a proposal to alter the design so that the output of the barrel shifter is connected to one input of the ALU instead of directly feeding both inputs from the register file.

(a) Draw diagrams of the two proposed layouts of the datapath. Identify differences in the control signals needed to drive the datapath, and how they would be set in each case for (i) an add instruction, and (ii) a left-shift instruction with a constant shift. [5 marks]

(b) The barrel shifter is capable of shifting its input to the left by any distance from 0 to 31 bits. Show a design for the barrel shifter in terms of multiplexers, and explain how it is able to shift by any number of bits. Suggest a simple way of constructing one of the multiplexers from logic gates. [5 marks]

(c) The processor designer notes, "supplying the immediate field from the instruction as a possible input to the shifter allows the addressing range of `sp`-relative load and store instructions to be extended." Explain the use of `sp`-relative addressing, the meaning of this observation, and how the modified design allows the improvement it suggests. [4 marks]

(d) Give an additional example of an addressing mode that could be provided with the new datapath but could not easily be provided before, and explain how the new design makes it possible. [3 marks]

(e) What effect is the change in design likely to have on the clock rate that the datapath can support, and why? [3 marks]

19 This question concerns implementations of the instruction set of a little-endian RISC-like machine. There are loads and stores with addressing modes where the address is computed either as the sum of two registers, or as the sum of a register and a small constant.

- The instructions `ldr` and `str` permit loads and stores where a 32-bit value is transferred between a register and memory.

- The instructions `ldrb` and `strb` load and store an 8-bit value in the least significant bits of a 32-bit register, with zero extension for the load.

For `ldr` and `str`, the effective address must be a multiple of 4, but `ldrb` and `strb` allow any address to be used.

The instruction set is implemented on a datapath that executes each instruction in a single cycle, and has separate memory interfaces for instructions and data.

(a) Give examples of constructs in a high-level language that can take advantage of the two addressing modes. [2 marks]

(b) Draw in outline and explain the design of a datapath that can implement 32-bit loads and stores using these addressing modes, stating what role is played by each functional unit in executing the instructions. [5 marks]

(c) If the `ldrb` instruction were not implemented, could the instructions

```
ldrb r0, [r1, #5]
```

and

```
strb r0, [r1, #5]
```

be replaced by an equivalent sequence of instructions that does not use `ldrb` or `strb` but only the `ldr` and `str` instructions? Assume that plenty of spare registers are available, and that the equivalent sequence may have a different effect on the condition codes. Either show a suitable sequence of instructions, or explain why one does not exist. [6 marks]

(d) A datapath that supports 8-bit loads might use an unchanged interface to data memory, fetch a whole word, but select the appropriate byte to write to a register. Describe a suitable functional unit that might be added to the datapath for this purpose, explaining what control signals it would use and showing how the unit might be constructed from logic gates. [5 marks]

(e) Could an additional modification also support 8-bit stores with the same memory interface, executing each store in a single cycle? Justify your answer. [2 marks]