SCHOOL OF MATHEMATICAL AND COMPUTER SCIENCES

Computer Science

F28HS

HARDWARE-SOFTWARE INTERFACE

MOCK EXAM

Duration: Two Hours

ANSWER THREE QUESTIONS

Answer each question in a separate script book.

1. In the course we discussed the importance of an architecture's *memory hierarchy* for the performance of a program.

(a) When a modern processor performs a load instruction, e.g. an `LDR` on the ARM processor, the system goes through several steps in order to retrieve the data from the appropriate level in the memory hierarchy. Discuss the main steps. (6)

(b) Modern architectures arrange the available memory in a "memory hierarchy" to profit from fast access time on-chip and large memory space off-chip. Answer the following questions:

- Is the hardware-cache directly addressable from the user program?
- Does the hardware-cache represent a local copy of the data in (parts of) the main memory?
- What's roughly the difference in access time between cache and main memory: $1\times$, $5\times$, $10\times$

(6)

(c) When a program loads the data of a memory location, does it load exactly one word? Explain your answer. What's the impact of this data loading on data access of neighbouring memory locations. (4)

(d) The following function takes two points as input and performs a (vector) addition on two input points, producing a new point as a result, if both input points are represented as cartesian coordinates. The representation of a point is specified in the type `location_t`: it can be either in cartesian form, with `x` and `y` coordinates, or in polar form, with magnitude `r` and angle `p`. Assume that both arguments of the function are in succeeding memory locations.

```
typedef struct cart_  {int x;  int y;} cart_coord_t;
typedef struct polar_ {int r;  int p;} polar_coord_t;
typedef union node {cart_coord_t c; polar_coord_t p; } either_t;

typedef struct {
  either_t val;
  coordinate_t ty;
} location_t;

static inline location_t *add_points_cart(location_t *pt1,
                                          location_t *pt2) {
  location_t *pt;
  if ((pt1->ty == pt2->ty) && (pt1->ty == cartesian)) {
    pt = (location_t *)malloc(sizeof(location_t));
    pt->val.c.x = pt1->val.c.x + pt2->val.c.x;
    pt->val.c.y = pt1->val.c.y + pt2->val.c.y;
    pt->ty = pt1->ty;
    return pt;
  } else {
    fprintf(stderr, "Illegal type of coordinates; should be cartesian
    return NULL;
  }
}
```

Answer the following questions:

1. What is the size (in bytes) of the `location_t` structure?
2. The RPi2 has a cache-line size of 64 bytes. How many memory lookups, as opposed to cache lookups, have to be taken when executing the function for two points?

(4)

(e) What's the title and the artist of the theme song for the course? (0)

2. In the course we discussed in detail how the GPIO pins on a Raspberry Pi 2 can be used to program external devices.

(a) You want to compose a simple C program, running under Raspbian on the RPi2, to turn on an LED that is attached to GPIO pin `pin`. From the technical manual of the ARM Cortex-A7 processor, you pick-up the base address of the GPIO registers and you correctly pick up the register and bit position. You correctly compose a a C program that writes into the correct register and bit position, however the program stops with an 'Illegal instruction' message. Explain what's going wrong. (6)

(b) You have a running application of an button-controlled-LED, as in the sample code from the tutorials, that uses pin 24 for the LED output and pin 23 for the button input. You are asked to change the program, so that the LED is connect through pin 16 instead. Assume that the wiring of the RPi2 does have an LED connected to pin 16. Answer the following questions.

To set the mode of the pin, you need to:

1. Change both the register and the bit position that is written to.
2. Change only register that is written to.
3. Change only the bit position that is written to.
4. Change nothing.

To turn on the LED on the new pin 16, you need to:

1. Change both the register and the bit position that is written to.
2. Change only register that is written to.
3. Change only the bit position that is written to.
4. Change nothing.

Having made the above changes, the modified program should work in using pin 16 instead:

1. True
2. False

(6)

(c) The following C code snippet aims to set the mode of GPIO pin 23 to 1 (output), so that it can be used to control an LED.

```
// setting the mode for PIN 23
fSel = 1; // register for setting mode of GPIO 23
```

```
shift = $\boxed{2}$; // bits in the register, corresponding to
GPIO 23
// C version of setting LED
(gpio + fSel) = $\boxed{3}$;// Sets bits to one = output
```

Fill in the blank sections to select the register to access $\boxed{1}$, the shift value $\boxed{2}$, and the code to compose the new value for the register $\boxed{3}$ (see the attached DATA SHEET). (8)

END OF PAPER

REGISTER TABLE

Address	Field Name	Description	Size	Read/ Write
0x 7E20 0000	GPFSEL0	GPIO Function Select 0	32	R/W
0x 7E20 0000	GPFSEL0	GPIO Function Select 0	32	R/W
0x 7E20 0004	GPFSEL1	GPIO Function Select 1	32	R/W
0x 7E20 0008	GPFSEL2	GPIO Function Select 2	32	R/W
0x 7E20 000C	GPFSEL3	GPIO Function Select 3	32	R/W
0x 7E20 0010	GPFSEL4	GPIO Function Select 4	32	R/W
0x 7E20 0014	GPFSEL5	GPIO Function Select 5	32	R/W
0x 7E20 0018	-	Reserved	-	-
0x 7E20 001C	GPSET0	GPIO Pin Output Set 0	32	W
0x 7E20 0020	GPSET1	GPIO Pin Output Set 1	32	W
0x 7E20 0024	-	Reserved	-	-
0x 7E20 0028	GPCLR0	GPIO Pin Output Clear 0	32	W
0x 7E20 002C	GPCLR1	GPIO Pin Output Clear 1	32	W
0x 7E20 0030	-	Reserved	-	-