EXAM
(Tentamen)

TDDI11
Embedded Software

2020-06-04 kl: 08-12

On-call (jour):
Ahmed Rezine: email (ahmed.rezine@liu.se), phone (013 - 28 1938).

Admitted material:
- You may access your individual notes, manuals, books, and even search the internet.
- No contacts, whether physical or virtual, are allowed during the duration of the exam with any person, whether the person is related to the course or not, except for contacting the examiner via email for clarifications if needed.
- Any **suspected breach** will be **systematically reported to the disciplinary board**. We will use Urkund after you submit via Lisam (Urkund is an automatic tool to combat plagiarism).

General instructions:
- **The questions may refer to your D_i digit, where i *in* $\{1, 2, 3\}$. The sequence $D_1D_2D_3$ corresponds to the last three digits of the** "anonymous-ID" you are assigned during the exam. For instance, if your "anonymous-ID" is A-2709 or A-86709 then D_1 is 7, D_2 is 0 and D_3 is 9 (all in base 10). In this case, $(D_1 + 2)$ is (7 + 2) which is 9, D_1D_2 is 70 and D_2D_3 is 09. Ask the examiner if this is unclear.
- The questions use $\lfloor . \rfloor$ for the floor function and $\lceil . \rceil$ for the ceil function. For instance, $\lfloor 1.5 \rfloor$ is 1 and $\lceil 1.5 \rceil$ is 2.
- The problems are **not ordered** according to difficulty. You are encouraged to read all problems carefully and completely before you begin.
- You may answer in either English or Swedish.
- Do **not take pictures of your solutions or draw them**. We only accept PDF files obtained from a text editor or a word processor. You are free to use any text editor (examples include notepad, vim, gedit, emacs, etc) or other text-based office programs (Microsoft Word or Open/Libre Office). We expect you to generate and to submit, via Lisam, a single PDF with **ordered answers.**
- Be **precise** in your statements and clearly **motivate** all statements and reasoning.
- **Explain** calculations and solution procedures.
- If in doubt about the question, write down your interpretation and assumptions.
- Grading: U, 3, 4, 5. The **preliminary** grading thresholds for p points are: $0 \leq p < 20$: U, $20 \leq p \leq 30$: 3, $31 \leq p \leq 35$: 4 and $36 \leq p \leq 40$: 5.

Problem 1. (13 points)
Clearly formulate your answers. Points will be withdrawn for incorrect/ambiguous justifications:

a. Assume UART based transmission. What is a parity bit? Describe a scenario (and explain) how this bit can be useful when transmitting the value $D_1 + D_2 + D_3$. For instance, if $D_1D_2D_3$ is 709, then $D_1 + D_2 + D_3$ is the decimal number 16. Describe a scenario where the parity bit does not help. (3pts)

b. Give an example of an embedded system that is also a hard-real time system. Why does it qualify to be an embedded system? What does it mean for it to be a hard-real time system? (3pts)

c. Write a C function "int check(char c)" that takes a char "c" as input (written as the sequence of 8 bits $c_7c_6c_5c_4c_3c_2c_1c_0$to clarify the question). The input could be the status of some peripheral. The function should return true exactly when the bit c_i (with i equal to $(\lfloor\frac{D_2}{2}\rfloor + 2)$) is 1. For instance, if your D_2 is 5 then $(\lfloor\frac{D_2}{2}\rfloor + 2)$ is 4 and your "check" method should return true exactly when c_4 is 1. (2 pts)

d. Give an example of an embedded system and use it to explain what inconsistent requirements mean and describe a pair of inconsistent requirements that might arise when designing your system. (2 pts)

e. Assume a memory mapped display at address 0xA4000. The sequence

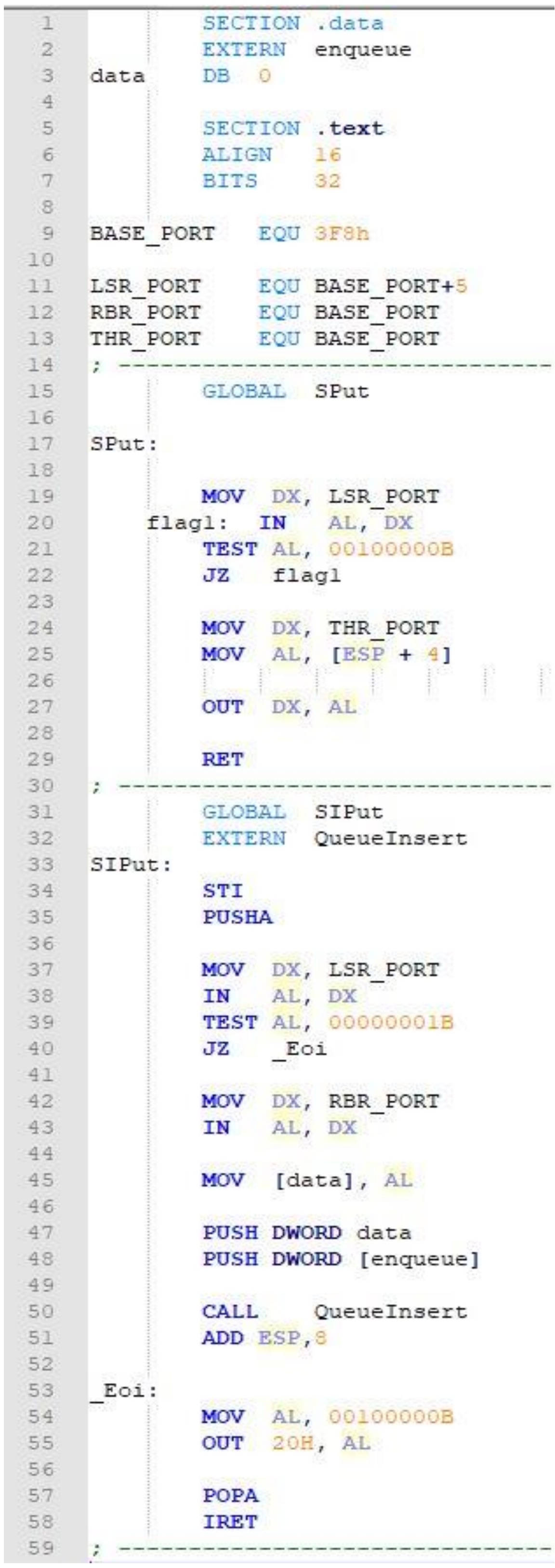
```
{
0x47, 0x46, 0x6f, 0x73, 0x6f, 0x7f, 0x64, 0x6c, 0x20, 0x4d,
0x6c, 0x2b, 0x75, 0x7b, 0x63, 0x46, 0x6b, 0x42, 0x21, 0x78,
0x2a, 0x68, 0x2a, 0x1d, 0x2a, 0x5a, 0x2a, 0x63, 0x2a, 0x1f,
0x47, 0x46, 0x6f, 0x73, 0x6f, 0x7f, 0x64, 0x6c, 0x20, 0x4d,
0x6c, 0x2b, 0x75, 0x7b, 0x63, 0x46, 0x6b, 0x42, 0x21, 0x78,
0x2a, 0x68, 0x2a, 0x1d, 0x2a, 0x5a, 0x2a, 0x63, 0x2a, 0x1f,
0x47, 0x46, 0x6f, 0x73, 0x6f, 0x7f, 0x64, 0x6c, 0x20, 0x4d,
0x6c, 0x2b, 0x75, 0x7b, 0x63, 0x46, 0x6b, 0x42, 0x21, 0x78,
0x2a, 0x68, 0x2a, 0x1d, 0x2a, 0x5a, 0x2a, 0x63, 0x2a, 0x1f,
0x47, 0x46, 0x6f, 0x73, 0x6f, 0x7f, 0x64, 0x6c, 0x20, 0x4d
}
```

consists of 100 bytes and is stored sequentially starting with byte 0x47 at address 0xA4000, 0x46 at address 0x4001, etc. The display has 10 rows with 5 characters on each row. Each character is encoded with two successive bytes: the first byte is the ASCII code of the character (the ASCII code of the first character is 0x47) and the second byte is an encoding of the foreground and background colors of the character (the encoding of the colors of the first character is 0x46).

1. Write a function "char* foo(int col)" that returns the address of the byte representing the ASCII code of the character at row D_1 and column col. For instance, if your $D_1$is 9 then foo(int col) should return the address of the ASCII byte of the character col on row 9. (2pts)
2. What is the result of foo($\lfloor D_2/2\rfloor$)? For instance, if your D_2 is 7, then the question is about the result of foo(3). (1pt)

Problem 2. (7 points)
Consider the code depicted in Figure 1 with the two methods SPut and SIPut. Answer the following questions:

1. Lines 20 and 27 use "IN" and "OUT" instead of "MOV". Explain the difference between the two approaches: using "MOV" versus using "IN" and "OUT". Describe advantages and disadvantages of each of the two approaches. (2pts)
2. "SPut" has a loop. "SIPut" does not have a loop. Both communicate with a peripheral. Explain what the difference between the two approaches is and describe advantages and disadvantages of each of the two approaches. (3pts)
3. Line 35 uses the instruction "PUSHA" to save all general-purpose registers onto the stack and line 57 retrieves them. Why is this performed? Suppose we did not have line 50 (i.e. no call to another method), would you still need to do a "PUSHA" in the beginning and a "POPA" at the end? Explain. (2pts)



```
 1              SECTION .data
 2              EXTERN  enqueue
 3  data        DB  0
 4
 5              SECTION .text
 6              ALIGN   16
 7              BITS    32
 8
 9  BASE_PORT   EQU 3F8h
10
11  LSR_PORT    EQU BASE_PORT+5
12  RBR_PORT    EQU BASE_PORT
13  THR_PORT    EQU BASE_PORT
14  ; ------------------------------
15              GLOBAL  SPut
16
17  SPut:
18
19              MOV  DX, LSR_PORT
20      flag1:  IN   AL, DX
21              TEST AL, 00100000B
22              JZ   flag1
23
24              MOV  DX, THR_PORT
25              MOV  AL, [ESP + 4]
26
27              OUT  DX, AL
28
29              RET
30  ; ------------------------------
31              GLOBAL  SIPut
32              EXTERN  QueueInsert
33  SIPut:
34              STI
35              PUSHA
36
37              MOV  DX, LSR_PORT
38              IN   AL, DX
39              TEST AL, 00000001B
40              JZ   _Eoi
41
42              MOV  DX, RBR_PORT
43              IN   AL, DX
44
45              MOV  [data], AL
46
47              PUSH DWORD data
48              PUSH DWORD [enqueue]
49
50              CALL    QueueInsert
51              ADD ESP,8
52
53  _Eoi:
54              MOV  AL, 00100000B
55              OUT  20H, AL
56
57              POPA
58              IRET
59  ; ------------------------------
```

Figure 1. Code snippet for problem 2.

Problem 3. (7 points)
Consider a task set with three periodic tasks: task 1 with execution time $C_1 = \lceil\frac{D_1}{2}\rceil$, task 2 with execution time $C_2 = \lceil\frac{D_2}{2}\rceil$, and task 3 with execution time $C_3 = \lceil\frac{D_3}{2}\rceil$. Observe the execution times can only be in the range 1 to 5. For instance, if your Anonymous-ID ends with 097, then $C_1 = 1, C_2 = 5$ and $C_3 = 4$. In this problem, you will have to find values for the periods of the tasks. The periods must be chosen among the values $\{2,4,8,12,24\}$. Of course, the period of each task must be larger or equal to the execution time of the task. Indeed, tasks and periods corresponding to red cells in Table 1 are impossible to schedule because they have a utilization that is strictly larger than 1. All three tasks are to run on the same sequential micro-controller (i.e. single processor) using some scheduling algorithm. Deadlines are as long as the periods.

1. Choose periods p_1, p_2, p_3 for tasks 1, 2 and 3 respectively, such that the utilization of the microcontroller (assuming these are the only tasks running) is larger than (or equal to) 70% and smaller than (or equal to) 90%. (2pts)
2. Which task would get the highest priority if Rate Monotonic Scheduling (RMS) is used? (1pt)

3. Can the tasks be scheduled using preemptive RMS? Explain using a table. The table should state which task is running at each time tick from 1 to the least common multiplier of p_1, p_2 and p_3. Observe the least common multiplier is at most 24 but may be smaller depending on your Anonymous-ID. (2pts)
4. Can the tasks be scheduled using preemptive Earliest Deadline First (EDF)? Explain by describing, in a table stating which task is running at each tick (again from 1 to the least common multiplier of p_1, p_2 and p_3). (2pts)

Table 1. Utilizations of individual tasks based on their execution times C and periods T

C \ T	2	4	8	12	24
1	0,50	0,25	0,13	0,08	0,04
2	1,00	0,50	0,25	0,17	0,08
3	1,50	0,75	0,38	0,25	0,13
4	2,00	1,00	0,50	0,33	0,17
5	2,50	1,25	0,63	0,42	0,21

Problem 4. (6 points)
The Mealy machine described in Table 2 takes sequences of 0s and 1s as input. It outputs 1 exactly when the last three inputs (including overlap) build the sequence 101. For instance, the machine in Table 2 outputs the sequence "00101010001" when it reads "10101011101".

You are asked to build a Mealy machine (describe it using a similar table where s_0 is the initial state, more states may be required) that outputs 1 exactly when the last 4 inputs build the sequence corresponding to the encoding of $((D_2\%3)+1)$ followed by $((D_3\%3)+1)$ followed by enough 0s to get to a sequence of 4 bits. Table 3 gives examples of binary sequences for several combinations of D_2 and D_3.

Table 2. Mealy machine outputs 1 when the last three inputs have been the sequence 101.

	in: 0	in: 1
s_0 (initial)	$\boldsymbol{out}: \mathbf{0}\ /\ \boldsymbol{goto}: \boldsymbol{s_0}$	$\boldsymbol{out}: \mathbf{0}/\ \boldsymbol{goto}: \boldsymbol{s_1}$
$\boldsymbol{s_1}$	$\boldsymbol{out}: \mathbf{0}/\ \boldsymbol{goto}: \boldsymbol{s_2}$	$\boldsymbol{out}: \mathbf{0}/\ \boldsymbol{goto}: \boldsymbol{s_1}$
$\boldsymbol{s_2}$	$\boldsymbol{out}: \mathbf{0}/\ \boldsymbol{goto}: \boldsymbol{s_0}$	$\boldsymbol{out}: \mathbf{1}/\ \boldsymbol{goto}: \boldsymbol{s_3}$
$\boldsymbol{s_3}$	$\boldsymbol{out}: \mathbf{0}/\ \boldsymbol{goto}: \boldsymbol{s_2}$	$\boldsymbol{out}: \mathbf{0}/\ \boldsymbol{goto}: \boldsymbol{s_1}$

Table 3. Obtaining the 4 bits sequence from $D2$ and $D3$

$\boldsymbol{D_2}$	$\boldsymbol{D_3}$	$\big((D_2\%3)+1\big)$ **then** $((D_3\%3)+1)$	**Resulting 4 bits sequence**
$D_2 = 0$	$D_3 = 1$	1 followed by 2	1 10 0

$D_2 = 3$	$D_3 = 9$	1 followed by 1	1 1 00
$D_2 = 4$	$D_3 = 6$	2 followed by 1	10 1 0
$D_2 = 7$	$D_3 = 2$	2 followed by 3	10 11

Problem 5. (7 points)
As it is often the case, newly pushed stack elements get smaller addresses. A function with one argument has just been called. The stack looks as follows:

```
byte        |     address
-----------------------
            |     0x3fffffdc
            |     0x3fffffdd
            |     0x3fffffde
            |     0x3fffffdf
            |     0x3fffffe0
            |     0x3fffffe1
            |     0x3fffffe2
            |     0x3fffffe3
            |     0x3fffffe4
            |     0x3fffffe5
            |     0x3fffffe6
            |     0x3fffffe7
            |     0x3fffffe8 <-- esp
            |     0x3fffffe9
            |     0x3fffffea
            |     0x3fffffeb
            |     0x3fffffec <-- (X)
            |     0x3fffffed
            |     0x3fffffee
            |     0x3fffffef
            |     0x3ffffff0 <-- (Y)
            |     0x3ffffff1
            |     0x3ffffff2
            |     0x3ffffff3
            |     0x3ffffff4 <-- (Z)
            |     0x3ffffff5
            |     0x3ffffff6
            |     0x3ffffff7
            |     0x3ffffff8 <-- ebp
            |     0x3ffffff9
            |     0x3ffffffa
            |     0x3ffffffb
```

Answer the following questions:

1. The register esp contains the value: “`0x3fffffe8`”. Explain what role is played by the 4 bytes stored at the addresses “0x3fffffe8, 0x3fffffe9, 0x3fffffea, 0x3fffffeb”. Why are they stored on the stack? (2pts)

2. The two first instructions of the function are:

```
push    ebp
mov     ebp, esp
```

   How do these two instructions update the stack and esp? How come these two instructions are often used at the beginning of functions? (2pts)

3. Explain Endianness. Suppose the argument to the function is the 32 bits integer corresponding to the sequence (in decreasing order of significance) of 4 bytes with the most significant byte containing the value 2 (written 0x2), followed by a byte containing the value D_2 (if your D_2 is 9 then the second byte is 0x9), then a byte containing the value 3 and finally (the least significant byte) containing the value D_3. Give the address in the stack of each one of these 4 bytes (3pts.)