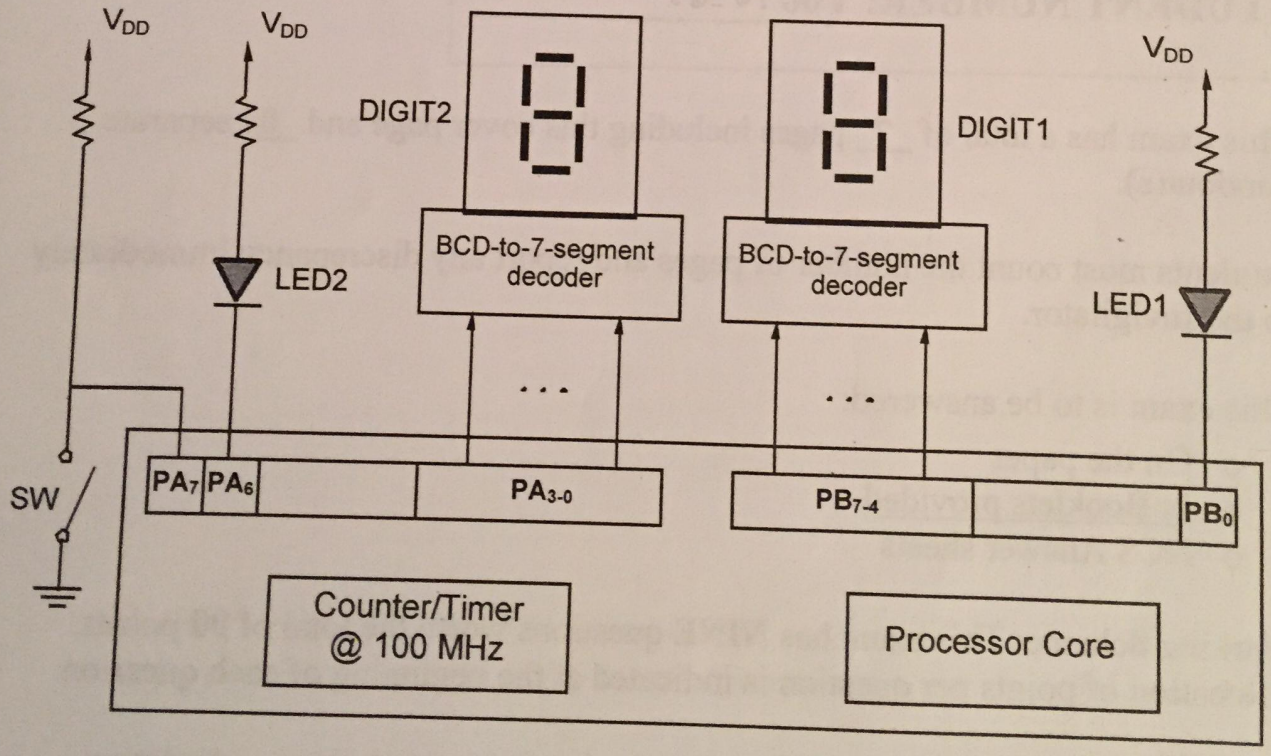
1. *[15 points]* The textbook's microcontroller is used in a system below and is responsible for two tasks: (1) incrementing either **DIGIT1** (if **LED1** is on) or **DIGIT2** (if **LED2** is on) every second, and (2) alternating between **LED1** and **LED2** being on, whenever the **SW** key is hit (i.e., pressed and then released). Write the corresponding C program, assuming that the **first task** is an ISR, whose address is stored at location **0x20**, and the **second task** is the main program. Also, assume that bit **6** of the processor status register (i.e., **PSR[6]**) is the processor's interrupt-enable bit, and **Ports A** and **B** are always ready to be accessed by the processor. Initially, **LED1** is on, **LED2** is off, and both **DIGIT1** and **DIGIT2** show **0**.

• *Main Program*: Every time the **SW** key is hit, i.e., pressed and then released (**PA_7** must first become 0 and then 1 again), **LED1** and **LED2** must swap their states: if **LED1** is on and **LED2** is off, then **LED1** becomes off and **LED2** becomes on, and vice versa. (Note: **LED1** and **LED2** are never both on, or both off.)

• *ISR*: The 100-MHz Counter/Timer must be configured to generate interrupts every second, and its ISR must increment **DIGIT1** if **LED1** is on, or increment **DIGIT2** if **LED2** is on. (Note: incrementing **9** gives **0**.)



2. *[20 points]* Consider a byte-addressable computer with 4-KB main memory and 256-byte cache having **eight blocks**, where each block consists of **eight 32-bit words**. Assume that the CPU reads 32-bit words from the following sequence of hexadecimal addresses:

088 094 100 10C 2F4 194 218 080 100 888

Show the cache contents (e.g., **[000]** = contents stored at address **000**) at the end of this sequence (10 addresses) and calculate the corresponding miss rate given that:

(a) Cache is direct-mapped.

(b) Cache is 2-way set-associative (2 blocks per set) with LRU replacement.

(c) Cache is 4-way set-associative (4 blocks per set) with LRU replacement.

(d) Cache is fully-associative with LRU replacement.

3. *[15 points]* Consider a C code fragment below, working on a given square matrix `int X[N][N]` (stored row by row, i.e., in the row-major order), where `N = 512`:

```
for (i = 0; i < N; i++) {
      int sum = 0;
      for (j = 0; j < N; j++) {
            sum = sum + X[i][j];          /* sum elements of X */
            X[i][j] = sum;                /* update elements of X */
      }
      for (j = 0; j < N; j++) {
            X[i][j] = X[i][j] - sum/N;    /* adjust elements of X */
      }
}
```

Determine the X-related page fault rate in the following three cases: (1) the main memory uses **1-KB** paging with four pages allocated for X, (2) the main memory uses **2-KB** paging with two pages allocated for X, and (3) the main memory uses **4-KB** paging with only one page allocated for X. Initially, no part of X is in the main memory.

4. *[5 points]* Table below specifies **three** independent preemptive tasks to be executed by a single processor. Show the task schedule based on *Rate Monotonic* (**RM**) prioritization.

Task T_i	Period P_i	WCET C_i	Deadline D_i	Initial Delay ϕ_i
T1	30	10	30	0
T2	40	10	40	0
T3	60	20	60	0

5. *[5 points]* Consider a pipelined datapath consisting of five stages:

F – fetch the instruction from the memory,

D – decode the instruction and read the source register(s),

C – execute the ALU operation specified by the instruction,

M – execute the memory operation specified by the instruction,

W – write the result in the destination register.

Identify data hazards in the code below and insert NOP instructions where necessary:

```
ADD   #4, R0, R0        // R0 = R0 + 4
SUB   R5, R3, R1        // R1 = R5 - R3
ADD   #4, R5, R5        // R5 = R5 + 4
ADD   R0, R1, R2        // R2 = R0 + R1
MOV   (R0), (R5)        // MEMORY[R5] = MEMORY[R0]
ADD   #4, R0, R0        // R0 = R0 + 4
SUB   R4, R2, R4        // R4 = R4 - R2
ADD   #4, R5, R5        // R5 = R5 + 4
MOV   R0, (R0)          // MEMORY[R0] = R0
MOV   (R4), R3          // R3 = MEMORY[R4]
```

6. *[10 points]*

(a) Show **decimal** number **+33.25** in the 32-bit IEEE-754 floating-point format.

(b) Given two 32-bit IEEE-754 floating-point numbers **X** and **Y** below, calculate (in the binary format) **Z = X–Y**, and then convert **Z** to the decimal format:

X = 0 00111100 00000000000000000000000,
Y = 1 00111111 10000000000000000000000.

7. *[5 points]* Suppose some FSM has 3 inputs, internal **Ready**, external bus-grant **Grant**, and external bus-free **Free** signals, as well as 2 outputs, bus-request **Req** and bus-lock **Lock** signals. Show its Moore-type state diagram, assuming that the FSM implements the following bus protocol: (1) initially, the FSM outputs **Req = 0** and **Lock = 0** and waits for both **Ready** and **Free** to be asserted; (2) After receiving **Ready = 1** and **Free = 1**, the FSM outputs **Req = 1** and **Lock = 0** and waits for **Grant** to be asserted; (3) After receiving **Grant = 1**, provided that both **Ready** and **Free** still equal **1**, the FSM outputs **Req = 0** and **Lock = 1** and waits for **Ready** to become **0**; once **Ready = 0**, the FSM returns to step (1). If **Ready = 0** or **Free = 0** when **Grant = 1** is received, the FSM returns to step (1). The FSM ignores the **Grant** input in steps (1) and (3), and it ignores the **Free** input in step (3).