CS473-System programming for SOC's

Dr. Theo Kluter

19.12.2023

Name:	**Sciper:**
_____	
Room:	**Place nr.:**

Important: Your way to come to your solution needs to be clearly visible!

Question:	Points	Score
1	7	
2	7	
3	5	
4	11	
Total	30	

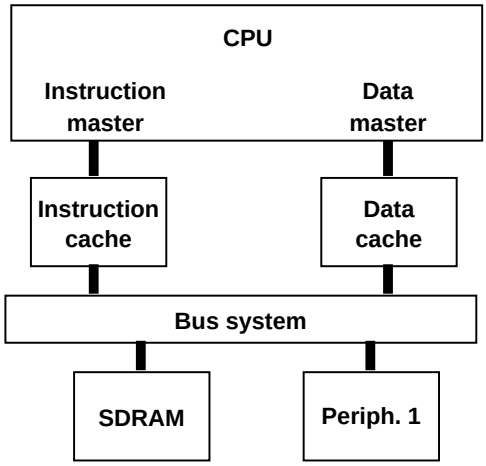
Question 1: *Theoretical questions* ..... (7 points)

(1point) (a) I read from a memory location the value `0x22110000`. However, I expect the value `0x00001122`. What is the problem that I forgot about?

(2points) (b) In an computer architecture we can have a *data-coherence* problem. Explain what this problem is and how it can occur in a single-cpu architecture.

(2points) (c) An embedded system can use caches and scratchpad memories. Explain the difference between the two.

Given is the block-diagram of an embedded system below:



(2points) (d) Give two different scenarios in which this system acts as a Harvard-computer architecture (you may draw in the figure).

Question 2: *Number formats* ..... (7 points)

In digital signal processing, we use very often *Finite Impulse Response (FIR)* filters. The algorithm of a FIR-filter is given by:

$$y[n] = b_0x[n] + b_1x[n-1] + ... + b_Nx[n-N] = \sum_{i=0}^{N} b_ix[n-i]$$

In this formula b_i are the filter coefficients, and $x[n-i]$ the samples coming from a sensor. We now that the samples coming from the sensor have the range $-4.0 \leq x[n] \leq 4.0$. Furthermore, with a good FIR-filter design, also the resulting values for $y[n]$ must be in the same range, hence: $-4.0 \leq y[n] \leq 4.0$. As a result, the filter coefficients b_i must be in the range: $-1.0 < b_i < 1.0$. The general function that calculates the result of the FIR-filter is given by:

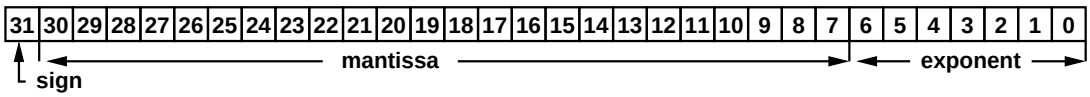
```
typedef struct SensorInfo {
  uint32_t writePointer;
  uint32_t readPointer;
  float sensorData[256];
} sensorInfoT;

float firFilter(float *coefficients, sensorInfoT *samples, unsigned int length) {
  float result = 0.0;
  for (unsigned int i = 0; i < length; i++)
    result += samples->sensorData[(samples->readPointer - i) & 0xFF] * coefficients[i];
  return result;
}
```

As you might imagine, this function is a critical function that needs to be optimized. Important in this algorithm is that we keep as much precision as we can. The CPU on which this algorithm is running has registers of 32-bits.

(2points) (a) We are going to transform this algorithm to a fixed-point implementation. Remember: we note the fixed-point format with `Qx.y`, where `x` is the number of bits before the decimal point, and `y` is the number of bits after the decimal point. What would be the best fixed-point representation for $sensorData[i]$, $coefficients[i]$, $result$, and the return value $y[n]$ of the function to keep the maximum on precision (note: (1) they do not have to be the same, and (2) we base the fixed-point version of $x[n]$, $y[n]$, and b_i on 32-bit)?

The fixed-point version is not precise enough for our filter. Hence, we decide to implement our own floating point version, with following definition:



where:

- $0.0 \leq mantissa < 1.0$ fixed point. Bit 30 is always 1 except for the value 0, where the mantissa is 0 and the exponent and sign can be anything.
- The exponent is excess-124.

The filter is rewritten as:

```
typedef myFloat uint32_t;

typedef struct SensorInfo {
  uint32_t writePointer;
  uint32_t readPointer;
  myFloat sensorData[256];
} sensorInfoT;

myFloat firFilter(myFloat *coefficients, sensorInfoT *samples, unsigned int length) {
  myFloat result = 0;
  for (unsigned int i = 0; i < length; i++)
    result = myAdd( result ,
      myMult(samples->sensorData[(samples->readPointer - i) & 0xFF], coefficients[i]) );
  return result;
}
```

(2points) (b) The above function calls two support function `myAdd()` and `myMult()`. This is a bad idea, better would be to in-line these two functions in the function `firFilter`, why (explain your answer)?

(3points) (c) Write the function `myMult()` that performs the multiplication. Hint: (1) as bit 30 is always 1, we only have to normalize if after the multiplication bit 31 is 1. (2) Underflow and overflow conditions can be ignored.

```
myFloat myMult(myFloat a, myFloat b) {
```

Question 3: *Hot-spot-detection* ..... (5 points)

We profiled a program and got following results:

- function 1 : 12365 cycles
- function 2 : 865312 cycles
- function 3 : 71392 cycles
- main : 1123 cycles
- run-time : 950192 cycles

With some simple rewrite we can speed-up function 3 by a factor of 8. As a result of this rewrite, function 2 reduces in execution time by 2%.

(1point) (a) How many cycles do the functions 2 and 3 execute after the rewrite?

- function 2 :
- function 3 :

(2points) (b) What is the total speed-up of the execution of this program due to the rewrite (explain your answer)?

(2points) (c) Name two reasons why function 2 is reduced in execution time due to only a rewrite of function 3:

Question 4: *Back-on-the-envelop* ..... (11 points)

We are given an embedded system with a CPU running at 50MHz. We want to attach a sensor to this system that generates an interrupt each time a new value is available. We are going to determine what the maximum sample rate of this sensor can be such that our system is still running. We start of with the assumption that our embedded system is ideal, meaning that we always have a hit in the instruction and data-caches.
It takes 3 CPU-cycles between the generation of an IRQ and the execution of the interrupt service routine. The interrupt service routine is given below:

```
sensorServiceRoutine:              #located at address 0x000000C8
   l.addi   r1,r1,-16
   l.sw     0x00(r1),r2
   l.sw     0x04(r1),r3
   l.sw     0x08(r1),r4
   l.sw     0x0C(r1),r5
   l.movhi  r2,hi(sensorBase)
   l.ori    r2,r2,lo(sensorBase)
   l.sw     4(r2),r0               #clear the interrupt
   l.lwz    r4,8(r2)               #read the sensor data
   l.movhi  r3,hi(sensorDataStruct)
   l.ori    r3,r3,lo(sensorDataStruct)
   l.lwz    r2,0(r3)               #get the offset in our buffer (writePointer)
   l.slli   r5,r2,2
   l.addi   r5,r5,r3
   l.addi   r5,r5,8               #r5 is the pointer to the buffer location
   l.addi   r2,r2,1
   l.andi   r2,r2,0xFF            #determine the next writePointer
   l.sw     0(r3),r2              #and update the structure
   l.sw     0(r5),r4              #save the sensor data in our buffer (sensorData)
   l.lwz    r2,0x00(r1)
   l.lwz    r3,0x04(r1)
   l.lwz    r4,0x08(r1)
   l.lwz    r5,0x0C(r1)
   l.rfe
   l.addi   r1,r1,16
```

The sensor data structure is given by:

```
typedef struct SensorInfo {
  uint32_t writePointer;
  uint32_t readPointer;
  uint32_t sensorData[256];
} sensorInfoT;

sensorInfoT sensorDataStruct;
```

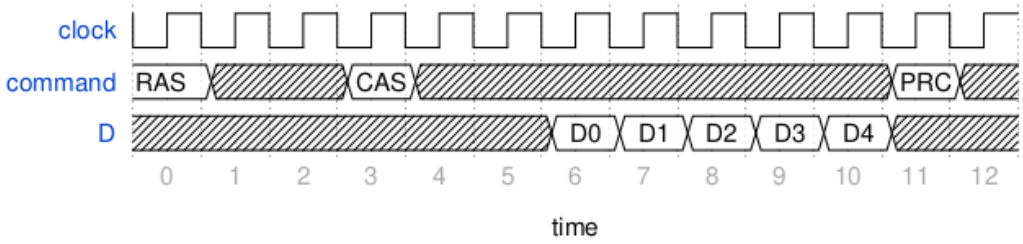
The sensor is mapped in the uncacheable region. It takes 15 CPU-cycles to read from or to write to the sensor. The moment the main program sees a difference between `sensorDataStruct.readPointer` and `sensorDataStruct.writePointer`, it will process the sensor data. The processing of the sensor data takes 1000 CPU-cycles, this includes one poll on the difference of `sensorDataStruct.readPointer` and `sensorDataStruct.writePointer`.
Our system has a direct mapped instruction cache of 1kByte with a cache-line size of 32 bytes. Furthermore it has a 2-way set associative data cache of 1kByte with a cache-line size of 32 bytes. Finally the system has a scratch-pad memory on the data side of 2 kBytes. The stack is located in the scratch-pad memory.

(3points) (a) Under the assumption that we only have hits in the data- and instruction cache, how much cycles does it take between the generation of an interrupt and the return to the main program (explain your answer)?

(2points) (b) Under the assumption that we only have hits in the data- and instruction cache, what would be the maximum sample-rate that our system can handle (explain your answer)?

(1point) (c) How many cache-lines are occupied by the `sensorServiceRoutine` in the instruction cache?

The program is stored in an external SDRAM. This SDRAM has a data-bus of 16 bits. The SDRAM is used in burst-mode. This SDRAM is connected by a controller to our bus-architecture that is 32-bit based. You can assume that we never hit the refresh-period of the SDRAM, and that our bus-architecture has 0-cycles access-time and latency. Furthermore, the SDRAM is running at the same clock as the CPU (hence 50MHz). A typical burst access for a burst-size of 5 is given by (where PRC is the pre-charge command):



(2points) (d) How many CPU-cycles does it take to load one instruction-cache line (explain your answer)?

(3points) (e) In the worst-case situation, the interrupt service routine aliases with our main program (hence they occupy the same cache-lines in the instruction cache). Given this situation, how much cycles are added to the execution of the interrupt service routine and the main program, and what would be the new maximum sample rate (explain your answer)?