Université d'Ottawa
Faculté de génie

École de science informatique et génie électrique

uOttawa
L'Université canadienne
Canada's university

University of Ottawa
Faculty of Engineering

School of Electrical Engineering and Computer Science

COURSE: CEG3136/CEG3536 **PROFESSORS:**

SEMESTER: **DATE:**
TIME:

Computer Architecture II

FINAL EXAMINATION

Solution

NAME and STUDENT NUMBER:_____/_____

Instructions:

- Answer ALL questions on the examination paper.
- This is a close-book examination**.**
- Use the provided space to answer the following questions. If more space is needed, use the back of the page.
- Show all your calculations to obtain full marks.
- Calculators are allowed.
- Read all the questions carefully before you start.

There are three (3) parts in this examination.

Part 1	**Short Answer (Theory)**	**20 marks**	
Part 2	**Assembler Prog.**	**5 marks**	
Part 3	**Measuring the Heart Rate**	**35 marks**	
Total		**60 marks**	

Total Pages: 11

Part 1 Short Answer Questions (Theory) (total 20 points)

Give a short answer to the following questions.

1. (2 points) Give the 2 control register used to configure the HCS12 parallel port A?

   DDRA (data direction register and PUCR (pullup control register)

2. (2 points) What is the longest time between overflow interrupts that can be configured on the HCS12 Timer assuming a 24 MHz system clock?
   128 (max scaling factor) * 41 2/3 ns (system clock period) * 65, 536 (number of ticks between two overflows) = 349.5 ms.

3. (4 points) The SCI (serial communications interface) has only one interrupt signal going to the CPU, which means only one interrupt service routine to service the SCI. Yet there are a number of sources for interrupts (transmission of a character complete, reception of a character, etc.). How can the source of the interrupt be determined? Provide two examples of such sources.

The status registers provides bits that indicate the action that caused the interrupt. Receive data register full (RDRF flag), overrun error (OR flag), Transmit data register full (TDRF flag), transmit complete (TC flag).

4. (2 points) What does the CPU12 use to find an interrupt service routine when an interrupt occurs?
   An interrupt vector from the interrupt vector table is associated to each interrupt and gives the addresses of the interrupt service routines.

5. (2 points) The ATD module can generate an interrupt after each conversion sequence. What is the maximum number of conversions in the conversion sequence?
   8

6. (2 points) What is the typical action taken to clear a flag when fast clear is NOT enabled?
   Write a 1 to the flag to reset it to 0.

7. (4 points) The HCS ATD module is configured as follows: ATDCTL2=0b11000010, ATDCTL3=0b00001000, ATDCTL4=0b01100101. Conversions are started by setting ATDCTL5 to 0b00100010 which initiates continuous conversions of the signal applied to 2. What is the maximum frequency in the signal applied to pin 2? (Assume a 24 MHz system clock).
   ADT clock cycles: 10 cycles for 10 bit conversion, 2+16 cycles for the sample time for a total of 28 ATD clock cycles per conversion
   ATD clock rate is 24 MHz/(2*(5+1)) = 2.0 MHz.
   Sampling rate will be 2.0 MHz / 28 (cycles/sample) = 71,428 samples/sec, i.e. 71.4 kHz
   Sampling rate is twice maximum frequency of signal, which must be 35.7 kHz.

8. (2 points) What are the 3 types of registers the CPU uses to interface with a hardware peripheral?
   Control register, status register, and data register.

Part 2 – Assembler Programming (total 5 points)

Translate the following C Pseudo-code to assembler.

C Function	Assembler Code

```
//Global Variables/Constants
int ix = 0;

/* ----------------------------
Function: addtoArray
Parameters: int val - a value
            to add to the array.
Description: Stores 16-bit
       value val in array using
       the global index, ix.
       ix is incremented so
       that the next call will
       add the value to the next
       element in the array.
       The index wraps to start
       when it reaches the end of
       the array.
-----------------------------*/
void addToArray(int val)
{
   array[ix] = val;  // add value
   ix++;  // increment index
   // Wrap around
   if(ix == LENGTH) ix = 0;
}
```

```
% GLOBAL Variables/Constants
LENGTH EQU 15
array DS.W LENGTH
ix DC.B 0

Subroutine: addToArray()
Parameters: val - in D register
Returns nothing.
Description:
   Stores 16-bit value (D) in array
   using using the index ix.
%-------------

addToArray
      pshy
      pshx
      pshd

      tfr d,y   ; save value of d, since use b
      ldx #array
      ldab ix
      sty b,x
      incb
      incb
      cpb #(2*LENGTH)
      bne endif
      clrb
endif:
      stab ix

      puld
      pulx
      puly
      rts
```

0.5 point for preserving values of used registers

1.5 point for storing val into proper location (here used index register and register b as offset, had do save d in y to table to used b register).

0.5 point for incrementing b (ix)

0.5 point for comparing B (ix) to LENGTH

1 point branch instruction

0.5 for clrb instruction.

0.5 for stab instruction.

Part 3 – System for Measuring the Heart Rate (35 points)

Your task is to develop the software module HeartRate that completes the development of a timer system used to measure the heart rate of a person. The system is designed to measure the heart rate using a finger sensor (placed on the end of a person's finger). See the exam annex for details on this system and guidelines to answer this part of the exam.

The following header file is available for HeartRate module.

```
/*--------------------------------
File: HeartRate.h

Description: Information for using the HeartRate Module
---------------------------------*/

// Definitions
#define TRUE 1
#define FALSE 0
#define NOT_AVAILABLE -1   // Heart Rate not available

// Bit Definitions for manipulating I/O register bits
#define BIT0 0b00000001
#define BIT1 0b00000010
#define BIT2 0b00000100
#define BIT3 0b00001000
#define BIT4 0b00010000
#define BIT5 0b00100000
#define BIT6 0b01000000
#define BIT7 0b10000000

// Prototypes
void initHeartRate(void); // Setups the hardware for measuring the heart rate
void startHRMeasurements(void);  // Starts up heart rate measurement
void stopHRMeasurements(void);  // Stops heart rate measurement
int getHeartRate(void);    // returns current heart rate.
```

The module file *HeartRate.c* starts with the following lines. Note that the global variables are not complete and local prototypes are not provided; do add any additional global variables and function prototypes required here.

```
/*-------------------------------------
File: HeartRate.c

Desription: Sofware module for measuring
the heart rate
-------------------------------------*/

// Include files
#include <stdtypes.h>
#include "mc9s12dg256.h"
#include "HeartRate.h"

// Add all additional global variables and local functions here.
// Definitions
#define NUM_SECONDS 15   // number of seconds to compute heart rate
#define FIFTY_MS_COUNT 37500  // At 1 1/3 micro-sec tick, need 37500 ticks for 50 ms
#define ONE_SEC_COUNT 20   // at 50 ms steps, require 20 to count to 1 second
#define FIFTY_MS_COUNT 37500  // At 1 1/3 micro-sec tick, need 37500 ticks for 50 ms
// Global variables for the module
static int count[NUM_SECONDS]; // count of heart beats each second for the last 15 sec
static int hrIx;               // index into array for adding count
static int available;      // set to TRUE if rate is availble and FALSE otherwise
static int heartRateCount;   // variable for storing count of heart beats
static int count50MilliSec;  // counts 50 ms steps to count to 1 second

// Local function prototypes
```

Marks for this section are assigned in each of the individual sections.

Question 1 (7 points)

Develop an initialisation function that initialises all hardware for measuring the heart rate.

This function does not start the timer channels for measuring the heart rate (for example, this function does not activate interruptions). Two other functions are used to start and stop heart rate measurements.

```
/*---------------------------
 * Function: initHeartRate()
 * Description:  Initialises
 *                  - Port B for activating and deactivating the circuit
 *                  - the timer channels for counting heart beats
*---------------------------*/
void initHeartRate()
{
    byte val;
    // Initiliase Port B, bit 0
    DDRB |= BIT0;  // Setup pin 0 as an output pin
    PORTB &= ~BIT0;  // clear bit to turn off heart rate circuit
    // Initialise Timer
    // Setup channel 2 as input capture on rising edges
    TIOS &= ~BIT2;
    // Set EDG2B (bit 5) and EDG2A (bit 4) to 00 - disconnect input pin
    TCTL4 &= ~(BIT5 | BIT4);
        // Setup Channel 3 for 1 second timing
    TIOS |= BIT3;
    TIE &= ~(BIT2|BIT3);  // turn off interrupts - started only when measurements
started
    // Indicate that heart rate is not available
    available = FALSE;
}
```

1 mark for initialising Port B.

1 mark for initialising TIOS for Channel 2 (input capture).

1 mark for initialising TCTL4.

1 mark for initialising TIOS for channel 3 – output compare..

1 mark for TIE.

1 mark for initialising available.

1 mark for declaring available as global variable – see page 5.

Question 2 (6 points)

Develop the function `startHRMeasurements()` which activates the circuit and configures the timer to measure the heart rate.

```
/*---------------------------
 * Function: startHRMeasurements()
 * Description: Starts measuring the heart rate.
 *---------------------------*/
void startHRMeasurements(void)
{
    // Turn on the circuit
    PORTB |= BIT0;
    // Start counting heart beats
    heartRateCount = 0;  // resets count to 0
    hrIx = 0;  // set index to start of array
    // Set EDG2B (bit 5) and EDG2A (bit 4) to 01 (rising edge)
    TCTL4 |= BIT4;
    TCTL4 &= ~BIT5;
    // Setup channel 3 for interrupts every second 50 ms (count FIFTY_MS_COUNT
ticks)
    TC3 += TCNT + FIFTY_MS_COUNT;
    // Enable interupts for both channels 2 and 3
    TIE |= BIT2 | BIT3;
}
```

1 mark for turning on circuit via Port B.

1 mark for initialising heartRateCount and hrIx.
1 mark for declaring these variables as global (page 5).

1 mark for initialising TCTL4.

1 mark for initialising TC3.

1 mark for TIE.

Question 3 (6 points)

Develop the function `stopHRMeasurements()` which deactivates the circuit and configures the timer to no longer measure the heart rate.

```
/*--------------------------
 * Function: stopHRMeasurements()
 * Description: Stops measuring the heart rate.
 *--------------------------*/
void stopHRMeasurements(void)
{
    // Turn off the circuit
    PORTB &= BIT1;
    // Turn off interrupts
    TIE &= ~(BIT2|BIT3);
    // Stop input-capture
    TCTL4 &= ~(BIT5 | BIT4);
    // Indicate that heart rate is not available
    available = FALSE;
}
```

1 mark turning off the circuit.

2 marks updating TIE

2 marks disabling input capture.

1 mark setting available to FALSE.

Question 4 (6 points)

Develop the function `getHeartRate()`, that returns the current heart rate (in beats/minute) if available, otherwise returns the value `NOT_AVAILABLE` (-1).

```
/*--------------------------
 * Function: getHeartRate()
 * Description: Returns the heart rate if available and NOT_AVAILABLE othewise.
 *--------------------------*/

int getHeartRate(void)
{
   int retVal = NOT_AVAILABLE;
   int total = 0;
   int i;
   if(available) // flag from ISR's
   {
      // Count all beats in count array
      for(i = 0; i< NUM_SECONDS; i++)
         total += count[i];
      retVal = 4*total;
   }
   return(retVal);
}
```

1 mark declarations.

1 mark for if statement

2 marks for summing heartbeats in array.

1 mark for multiplying by 4.

1 mark for returning value.

Question 5 (10 points)

Develop two ISR functions to service interrupts from two timer channels: one to count the heart beats, and the other to update the count array every second.

```
/*--------------------------
 * Function: ISR's
 * Description: Two ISRs used for measuring heart rate.
 *--------------------------*/

// ISR to count the heart beats
void interrupt VectorNumber_Vtimch2 countBeatsISR()
{
    // Input capture event on channel 2
    heartRateCount++;  // increment the count
    TFLG2 |= BIT2;  // clears C2F
}
// ISR to update the count array
void interrupt VectorNumber_Vtimch3 updateCountArray(void)
{
    TC3 += FIFTY_MS_COUNT; // setup to count for another 50 ms
                           // also clears TC3 interrupt and C3F
    count50MilliSec++;  // increment 50 ms count
    if(count50MilliSec == ONE_SEC_COUNT)
    {
       count50MilliSec = 0 ; // reset count
       heartRateCount = 0;  // save count in last sec and reset
       count[hrIx] = heartRateCount;
       hrIx = (hrIx+1) % NUM_SECONDS;  // reset the index
       if(!hrIx) available = TRUE;  // when hrIx reset to 0, then heart rate is
available
    }
}
```

1 mark increment heartRateCount.

1 mark for clearing the flag C2F

1 mark for updating TC3.

1 mark incrementing count50MilliSec and if statement.

1 mark reset count50MilliSec and heartRateCount to 0.

1 mark save heartRateCount in array.

1 mark to update hrIx.

1 mark to update available.

1 mark use of global constants (FIFTY_MS_COUNT, etc.) instead of magic numbers.

1 mark declaration of global variables on page 5: count50MilliSec, count, heartRateCount.