***King Fahd University of Petroleum and Minerals***

***College of Computer Science and Engineering***

***Computer Engineering Department***

**COE 306: INTRODUCTION TO EMBEDDED SYSTEMS**

**Term 161 (Fall 2016-2017)**

**Major Exam 1**

**Saturday Oct. 29, 2016**

**Time: 90 minutes, Total Pages: 10**

**Name:\_KEY\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ID:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Section: \_\_\_\_\_\_\_**

**Notes:**

* Do not open the exam book until instructed
* Answer all questions
* All steps must be shown
* Any assumptions made must be clearly stated

|  |  |  |
| --- | --- | --- |
| **Question** | **Max Points** | **Score** |
| **Q1** | **28** |  |
| **Q2** | **12** |  |
| **Q3** | **12** |  |
| **Q4** | **8** |  |
| **Total** | **60** |  |

Dr. Aiman El-Maleh

#  **[28 Points]**

# **(Q1)** Fill in the blank in each of the following questions:

## The difference between hard and soft real time deadlines is that missing hard deadlines causes failure while missing sot deadlines results in degraded performance.

## Microprocessors have higher flexibility and lower performance than ASICS.

## Requirements are plain language description of what the user wants and expects to get.

## An example of a non-functional requirement is time required to compute output, cost, size, weight, power consumption, reliability, etc.

## Specification is a formal more precise description of the system that reflects the customer’s requirements in a way that can be clearly followed during design.

## The difference between von Neumann architecture and Harvard architecture is that von Neumann architecture has one memory for data and program while Harvard architecture has separate memories for data and program and allows two simultaneous memory fetches.

## A superscalarprocessor uses specialized logic to identify at run time instructions that can be executed simultaneously while a VLIWprocessor relies on the compiler to determine what combinations of instructions can be legally executed together.

## In a RISC system with memory-mapped I/O, input/output operations are performed using load/store instructions.

## The PICmicro PIC16F has a Harvard (von Neumann /Harvard) architecture.

## The TI C55X DSP has a CISC (RISC /CISC) architecture.

## The C64x processor has the capability of executing up to eight instructions per cycle.

## The C55x processor has instructions to allow repeating the execution of a block of instructions.

## In an interrupt-based I/O system, a device knows that its request is accepted by seeing its priority number on the interrupt acknowledge lines.

## Given that two devices A and B are connected to a CPU through two interrupt lines with device A having higher priority than B. Suppose that the interrupt handler of device A executes in 30 cycles while that of B executes in 25 cycles. Assume that each instruction in the handlers executes in one clock cycle. If device A initiates an interrupt at the end of cycle 5 when the handler of device B is executing, then the handler of device B will finish execution by the end of cycle 55.

## If more than one device have the same priority and are connected to the same interrupt pin, then the interrupt handler will know the device who initiated the interrupt by checking the status register of each device.

## Interrupt Vectors allow an interrupting device to specify its handler.

## Two Cache memory organizations are fully-associative, direct-mapped, N-way set-associative.

## Given that **h** is the cache hit rate, **tcache** is the cache access time and **tmain** is the memory access time, then the average memory access time is **tavg = tcache + (1-h) tmain.**

## Two basic schemes for mapping logical addresses to physical addresses are: segmentation and paging.

**[12 Points]**

# **(Q2)**

## [6 points] Translate the given C code into ARM assembly code with **minimum** instructions:

**volatile** **static** **int** Array[10] = {75,20,50,40,55,60,10,85,100,90};

**int** Max=Array[0];

**int** Min=Array[0];

 **for** (**int** i=1; i<10; i++)

 **if** (Array[i]<Min)

 Min = Array[i];

 **else if** (Array[i] > Max)

 Max = Array[i];

#  adr r0, Array

#  ldr r1, [r0] ; min

#  ldr r2, [r0] ; max

#  mov r3, #1 ; i=1

# ForLoop

#  ldr r4, [r0, r3, lsl #2] ; get Array[i]

#  cmp r4, r1 ; if (Array[i]<Min)

#  movlt r1, r4 ; Min = Array[i]

#  blt Skip

#  cmp r4, r2 ; if (Array[i]>Max)

#  movgt r2, r4 ; Max = Array[i];

# Skip

#  add r3, r3, #1 ; i++

#  cmp r3, #9 ; i<10

#  bne ForLoop

# Array DCD 75,20,50,40,55,60,10,85,100,90

## [3 points] Write an ARM code fragment that multiplies the content of register r0 by 217 without the use of multiplication instructions with the minimum number of instructions. HINT: 217=31\*7.

rsb r0, r0, r0, lsl #5 ; r0 = r0\*31

 rsb r0, r0, r0, lsl #3 ; r0 = r0\*31\*7

## [3 points] Determine the content of register 0x27 after executing the following PIC16F assembly code:

MOVLW 0xA7

MOVWF 0x25

MOVLW 4

MOVWF 0x26

CLRF 0x27

 NEXT MOVF 0x25, w

ANDLW 3

ADDWF 0x27, f

RRF 0x25, f

RRF 0x25, f

DECFSZ 0x26

GOTO NEXT

This code scans the content of register 0x25 as a group of 2-bits and adds them up and stores the sum in register 0x27. So, the content of register 0x27 is 3+1+2+2=8.

**[12 Points]**

# **(Q3)** A system has two memory-mapped I/O devices. The first device has a 16-bit status register at address 0xA000, immediately followed by a 32-bit data register. The second device has an 8-bit status register at address 0xB000, followed by a 32-bit data register. The first device is used to receive data (i.e., input device). The most-significant bit in the status register is a *data ready flag*, which is set automatically by the device whenever new data is received. For the device to receive more data, the *data ready flag* must be manually reset by software to indicate that the current data have been processed.

# The second device is used to send data (i.e., output device). Bit 0 of its status register is a read-only *ready to send flag*, and bit 7 is a *transmit enable* command bit that is automatically reset by the device after each transmission.

# We would like to write software that collects 32-bit words of unsigned values received through the first device, and computes the maximum of received data until the second device becomes ready to send. Once the second device becomes ready to send data, the maximum word is sent using the second device. Once the maximum is sent, the maximum computation is restarted for the next sample of data, ignoring the previously received data samples.

* 1. Write a C program that implements this behavior using polling only.

#define DEV1\_STATUS 0xA000

#define DEV1\_DATA 0xA002

#define DEV2\_STATUS 0xB000

#define DEV2\_DATA 0xB001

int main(void) {

unsigned int max = 0; // holds max of data

 while(1) {

 if ((\* (unsigned short \*) DEV1\_STATUS) & (1<<15)) { // data ready flag is set

 if ( (\* (unsigned int \*) DEV1\_DATA) > max)

 max = (\* (unsigned int \*) DEV1\_DATA);

 (\* (unsigned short \*) DEV1\_STATUS) &= ~(1<<15); // reset data ready flag

 }

 if ((\* (char \*) DEV2\_STATUS) & 1) { // ready to send

 (\* (unsigned int \*) DEV2\_DATA) = max;

 (\* (char \*) DEV2\_STATUS) |= (1<<7); // transmit enable

 max = 0;

 }

 }

}

* 1. Assuming that each device has its own interrupt handler, write the handlers for each device in C. The first device generates an interrupt request upon receiving new data. The second device generates an interrupt request upon becoming ready to send new data.

Use the signatures:

**void** device1\_handler(**void**);

**void** device2\_handler(**void**);

unsigned int max = 0;

void device1\_handler(void) {

 if ( (\* (unsigned int \*) DEV1\_DATA) > max)

 max = (\* (unsigned int \*) DEV1\_DATA);

 (\* (unsigned short \*) DEV1\_STATUS) &= ~(1<<15); // reset data ready flag

}

void device2\_handler(void) {

 (\* (unsigned int \*) DEV2\_DATA) = max;

 (\* (char \*) DEV2\_STATUS) |= (1<<7); // transmit enable

 max = 0;

}

 **[8 Points]**

**(Q4)** In a virtual memory system, 20 bits are used to identify the page number, and 12 bits are used to specify the offset of an address within a page. The system supports up to 64 GB of physical memory.

1. How large is the virtual memory?

2(20+12) = 232 = 4 GB.

1. How wide are physical addresses?

log2 (64G) = 36 bits.

1. How many entries are there in the full flat page table?

220 = 1 M entries.

1. How wide is each entry of the page table for storing the physical page number?

36-12 = 24 bits.

1. Given the logical address 0x000100B8, what is the page number, in hexadecimal, for the page that contains this address? What is the offset of this address within its page (hexadecimal)?

Page Number is 0x000010, Offset is 0x0B8.

1. Suppose that the page of the logical address 0x000100B8 got mapped into physical page number 0xFF, what is the physical address corresponding to this logical address?

Physical address is 0x0000FF0B8.

1. If two-level page tables are used with the first-level page table having 1024 entries, how many entries will be in each of the second-level page tables?

1024 entries.

**ARM Instruction Set**

****

****

**PIC16 Instruction Set**

****