King Fahd University of Petroleum and Minerals
College of Computer Science and Engineering
Computer Engineering Department

COE 306: INTRODUCTION TO EMBEDDED SYSTEMS
Term 171 (Fall 2017-2018)
Major Exam 1
Saturday Oct. 28, 2017

Time: 120 minutes, Total Pages: 11

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	29	
Q2	12	
Q3	12	
Q4	9	
Total	62	

Dr. Aiman El-Maleh

[29 Points]

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

(1) The difference between a microprocessor and a microcontroller is that a microcontroller includes I/O devices and on-board memory.

(2) Three characteristics of embedded systems are sophisticated functionality, real-time operation, low manufacturing cost, low power and Designed to tight deadlines by small teams.

(3) Missing deadlines causing failure of an embedded system are called hard real time deadlines.

(4) Microprocessors have higher flexibility and lower performance than FPGAs.

(5) The embedded system design process has the following steps: Requirements, Specification, Architecture, Component design, System integration.

(6) Specification of an embedded system should be understandable, unambiguous and should not imply a particular architecture.

(7) Power consumption is an example of non-functional requirement.

(8) In UML, behavior could be described using state machines and sequence diagrams.

(9) Harvard architecture has separate memories for data and program, and allows two simultaneous memory fetches

(10) Two types of multiple instruction issue processors are superscalar and VLIW.

(11) In ARM processors, conditional execution of instruction allows very dense in-line code without branches.

(12) Given that r0=0x5 and r1=0x200, execution of the instruction str r0, [r1], #12 will store the value 0x5 at memory location 0x200 and the content of register r1 will be 0x20c.

(13) The PICmicro PIC16F has a 13-bit PC and 8-bit word size.

(14) In the PICmicro PIC16F, an 8-level stack is used for storing and restoring PC values when calling and returning from procedures.

(15) In the PICmicro PIC16F, indirect memory addressing is performed by taking an 8-bit indirect address from FSR and 1 bit, IRP, from the status register.

(16) In TI C55X DSP processor, the registers BRC0, RSA0 and REA0 are used for repeating the execution of a block of instructions where BRC0 stores the repetition count, RSA0 stores the starting address of the instruction block and REA0 stores the ending address of the instruction block.

(17) In TI C55X DSP processor, the registers BKC and BSAC are used for implementing a circular buffer in memory where BKC stores the circular buffer size and BSAC stores the starting address of the buffer.

(18) The TI C64X DSP processor has the capability of executing up to 8 instructions per cycle.

(19) A memory-mapped I/O means an address is assigned to each I/O and memory read/write instructions are used to communicate with I/O devices.

(20) In an interrupt-based I/O system with 8 priorities, the interrupt acknowledge signal has 3 bits.

(21) 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 B initiates an interrupt at the end of cycle 5 when the handler of device A is executing, then the handler of device A will finish execution by the end of cycle 30.

(22) Interrupt vectors allow an interrupting device to specify its handler by sending an interrupt vector which is used as an index to an interrupt vector table which stores the address of the handler.

(23) Two types of Cache misses are compulsory miss, conflict miss and capacity miss.

(24) Given two-level cache memory, L1 and L2, **h1** is L1 cache hit rate, **h2** is L2 cache hit rate, $\mathbf{t_{L1}}$ is the L1 cache access time, $\mathbf{t_{L2}}$ is the L2 cache access time and $\mathbf{t_{main}}$ is the memory access time, then the average memory access time is $t_{avg} = t_{L1} + (1-h_1)t_{L2} + (1-h_1)(1-h_2)t_{main}$.

(25) In memory segmentation, physical address is computed based on adding segment base address and logical address.

(26) Two methods of reducing power consumption are: reduce power supply voltage, run at lower clock frequency, disable function units with control signals when not in use, disconnect parts from power supply when not in use.

[12 Points]

(Q2)

(i) **[7 points]** Translate the given C code into ARM assembly code with **minimum** instructions:

```
volatile static int Array[10] = {75,60,55,40,85,60,90,88,100,70};

int cnt = 0;
for (i=0; i != 10; i++) {
  if ( (Array[i] > 70) && (Array[i] <= 100) ) cnt = cnt + 1;
}
```

```
            adr         r0, Array
            mov         r1, #0      ; count=0
            mov         r2, #0      ; i=0
ForLoop
            ldr         r3, [r0, r2, lsl #2]    ; get Array[i]
            cmp         r3, #70     ; if (Array[i]>70)
            ble         Skip
            cmp         r3, #100    ; if (Array[i]<=100)
            addle       r1, r1, #1  ; count = count+1;
Skip
            add         r2, r2, #1  ; i++
            cmp         r2, #10     ; i!=10
            bne         ForLoop
Array       DCD         75,60,55,40,85,60,90,88,100,70
```

(ii) [2 points] Write an ARM code fragment that multiplies the content of register r0 by 225 without the use of multiplication instructions with the minimum number of instructions. HINT: 225=15*15.

```
rsb        r0, r0, r0, lsl #4      ; r0 = r0*15
rsb        r0, r0, r0, lsl #4      ; r0 = r0*15*15
```

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

```
      MOVLW 0x85
      MOVWF 0x25
      MOVLW 8
      MOVWF 0x26
      CLRF  0x27
NEXT  BTFSS 0x25, 0
      INCF 0x27, f
      RRF   0x25, f
      DECFSZ 0x26
      GOTO  NEXT
```

This code counts the number of 0’s in register 0x25 and stores the count in register 0x27. So, the content of register 0x27 is 5.

[12 Points]

(Q3) A system has two memory-mapped I/O devices. The first device has an 8-bit status register at address 0xA000, immediately followed by a 32-bit data register. The second device has a 16-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 has 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 15 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 signed values received through the first device, and computes the average of received data until the second device becomes ready to send. Once the second device becomes ready to send data, the average word is sent using the second device. Once the average is sent, the average computation is restarted for the next sample of data, ignoring the previously received data samples.

(i) Write a C program that implements this behavior using polling only.

```
#define DEV1_STATUS 0xA000
#define DEV1_DATA 0xA001
#define DEV2_STATUS 0xB000
#define DEV2_DATA 0xB002
int main(void) {

int sum = 0; // holds sum of  data

int count = 0; // holds count of  data

  while(1) {
     if ((* (char *) DEV1_STATUS) & (1<<7)) { // data ready flag is set
         sum += (* (int *) DEV1_DATA);
         count++;
        (* (char *) DEV1_STATUS) &= ~(1<<7); // reset data ready flag
      }
      if ((* (unsigned short *) DEV2_STATUS) & 1) { // ready to send
        (* (int *) DEV2_DATA) = sum/count;
        (* (unsigned short *) DEV2_STATUS) |= (1<<15); // transmit
enable

        sum = 0; count=0;
      }
  }
}
```

(ii) 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**);

```
int sum = 0; // holds sum of  data

int count = 0; // holds count of  data


void device1_handler(void) {

    sum += (* (int *) DEV1_DATA);
    count++;
    (* (char *) DEV1_STATUS) &= ~(1<<7); // reset data ready flag

}


void device2_handler(void) {

    (* (int *) DEV2_DATA) = sum/count;
    (* (unsigned short *) DEV2_STATUS) |= (1<<15); // transmit enable
    sum = 0; count=0;

}
```

[9 Points]

(Q4) Given a virtual memory system with 32-bit logical addresses and 1K Byte pages. The system supports up to 1 GB of physical memory.

(i) How many bits **are** used for the page number and how many bits are used for the offset within a page?

Number of bits for the offset = 10 bits
Number of bits for the page number = 32-10 = 22 bits

(ii) How many entries are there in the full flat page table?

2^{22} = 4 M entries.

(iii) How wide is each entry of the page table for storing the physical page number?

30 -10 = 20 bits.

(iv) Given the logical address 0x00020FB8, 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 0x00083, Offset is 0x3B8.

(v) Suppose that the page of the logical address 0x00020FB8 got mapped into physical page number 0x20, what is the physical address corresponding to this logical address?

Physical address is 0x000083B8.

(vi) If two-level page tables are used with the first-level page table having 2048 entries, how many entries will be in each of the second-level page tables? How many page tables will be allocated for an 8 Mbyte program?

Number of entries in the second-level page tables is 2048 entries.
Number of needed pages is the first-level table and four second-level tables = 5 page tables.

ARM Instruction Set

Mnemonic	Instruction	Action
ADC	Add with carry	Rd := Rn + Op2 + Carry
ADD	Add	Rd := Rn + Op2
AND	AND	Rd := Rn AND Op2
B	Branch	R15 := address
BIC	Bit Clear	Rd := Rn AND NOT Op2
BL	Branch with Link	R14 := R15, R15 := address
BX	Branch and Exchange	R15 := Rn, T bit := Rn[0]
CDP	Coprocesor Data Processing	(Coprocessor-specific)
CMN	Compare Negative	CPSR flags := Rn + Op2
CMP	Compare	CPSR flags := Rn - Op2
EOR	Exclusive OR	Rd := (Rn AND NOT Op2) OR (op2 AND NOT Rn)
LDC	Load coprocessor from memory	Coprocessor load
LDM	Load multiple registers	Stack manipulation (Pop)
LDR	Load register from memory	Rd := (address)
MCR	Move CPU register to coprocessor register	cRn := rRn {<op>cRm}
MLA	Multiply Accumulate	Rd := (Rm * Rs) + Rn
MOV	Move register or constant	Rd : = Op2
MRC	Move from coprocessor register to CPU register	Rn := cRn {<op>cRm}
MRS	Move PSR status/flags to register	Rn := PSR
MSR	Move register to PSR status/flags	PSR := Rm
MUL	Multiply	Rd := Rm * Rs
MVN	Move negative register	Rd := 0xFFFFFFFF EOR Op2
ORR	OR	Rd := Rn OR Op2
RSB	Reverse Subtract	Rd := Op2 - Rn
RSC	Reverse Subtract with Carry	Rd := Op2 - Rn - 1 + Carry
SBC	Subtract with Carry	Rd := Rn - Op2 - 1 + Carry
STC	Store coprocessor register to memory	address := CRn
STM	Store Multiple	Stack manipulation (Push)
STR	Store register to memory	<address> := Rd
SUB	Subtract	Rd := Rn - Op2
SWI	Software Interrupt	OS call
SWP	Swap register with memory	Rd := [Rn], [Rn] := Rm
TEQ	Test bitwise equality	CPSR flags := Rn EOR Op2
TST	Test bits	CPSR flags := Rn AND Op2

PIC16 Instruction Set

Byte Oriented Operations		
addwf	f,d	Add W and f
andwf	f,d	AND W with f
clrf	f	Clear f
clrw	-	Clear W
comf	f,d	Complement f
decf	f,d	Decrement f
decfsz	f,d	Decrement f, Skip if 0
incf	f,d	Increment f
incfsz	f,d	Increment f, Skip if 0
iorwf	f,d	Inclusive OR W with f
movf	f,d	Move f
movwf	f	Move W to f
nop	-	No Operation
rlf	f,d	Rotate Left f through Carry
rrf	f,d	Rotate Right f through Carry
subwf	f,d	Subtract W from f
swapf	f,d	Swap nibbles in f
xorwf	f,d	Exclusive OR W with f

Bit Oriented Operations		
bcf	f,b	Bit Clear f
bsf	f,b	Bit Set f
btfsc	f,b	Bit Test f, Skip if Clear
btfss	f,b	Bit Test f, Skip if Set
Literal and Control Operations		
addlw	k	Add literal and W
andlw	k	AND literal with W
call	k	Call subroutine
clrwdt	-	Clear Watchdog Timer
goto	k	Go to address
iorlw	k	Inclusive OR literal with W
movlw	k	Move literal to W
retfie	-	Return from interrupt
retlw	k	Return with literal in W
return	-	Return from Subroutine
sleep	-	Go into standby mode
sublw	k	Subtract W from literal
xorlw	k	Exclusive OR literal with W