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:

(1) The difference between hard and soft real time deadlines is <u>that missing hard deadlines causes failure while missing sot deadlines results in degraded performance</u>.

(2) Microprocessors have higher <u>flexibility</u> and lower <u>performance</u> than ASICS.

(3) Requirements are plain language description of what the user wants and expects to get.

(4) An example of a non-functional requirement is <u>time required to compute output, cost, size, weight, power consumption, reliability, etc</u>.

(5) Specification is <u>a formal more precise description of the system that reflects the customer's requirements in a way that can be clearly followed during design.</u>

(6) The difference between von Neumann architecture and Harvard architecture is <u>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</u>.

(7) A <u>superscalar</u> processor uses specialized logic to identify at run time instructions that can be executed simultaneously while a <u>VLIW</u> processor relies on the compiler to determine what combinations of instructions can be legally executed together.

(8) In a RISC system with memory-mapped I/O, input/output operations are performed using <u>load/store</u> instructions.

(9) The PICmicro PIC16F has a <u>Harvard</u> (von Neumann /Harvard) architecture.

(10) The TI C55X DSP has a <u>CISC</u> (RISC /CISC) architecture.

(11) The C64x processor has the capability of executing up to eight instructions per cycle.

(12) The C55x processor has instructions to allow repeating the execution of a block of instructions.

(13) 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.

(14) 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.

(15) 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.

(16) Interrupt Vectors allow an interrupting device to specify its handler.

(17) Two Cache memory organizations are fully-associative, direct-mapped, N-way set-associative.

(18) Given that **h** is the cache hit rate, $\mathbf{t_{cache}}$ is the cache access time and $\mathbf{t_{main}}$ is the memory access time, then the average memory access time is $\mathbf{t_{avg} = t_{cache} + (1-h)\, t_{main}}$.

(19) Two basic schemes for mapping logical addresses to physical addresses are: segmentation and paging.

[12 Points]

(Q2)

(i) [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
```

(ii) [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
```

(iii) [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.

(a) 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;
                 }
           }
        }
```

(b) 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.

(a) How large is the virtual memory?

$2^{(20+12)} = 2^{32} = 4$ GB.

(b) How wide are physical addresses?

$\log_2 (64G) = 36$ bits.

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

$2^{20} = 1$ M entries.

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

36-12 = 24 bits.

(e) 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.

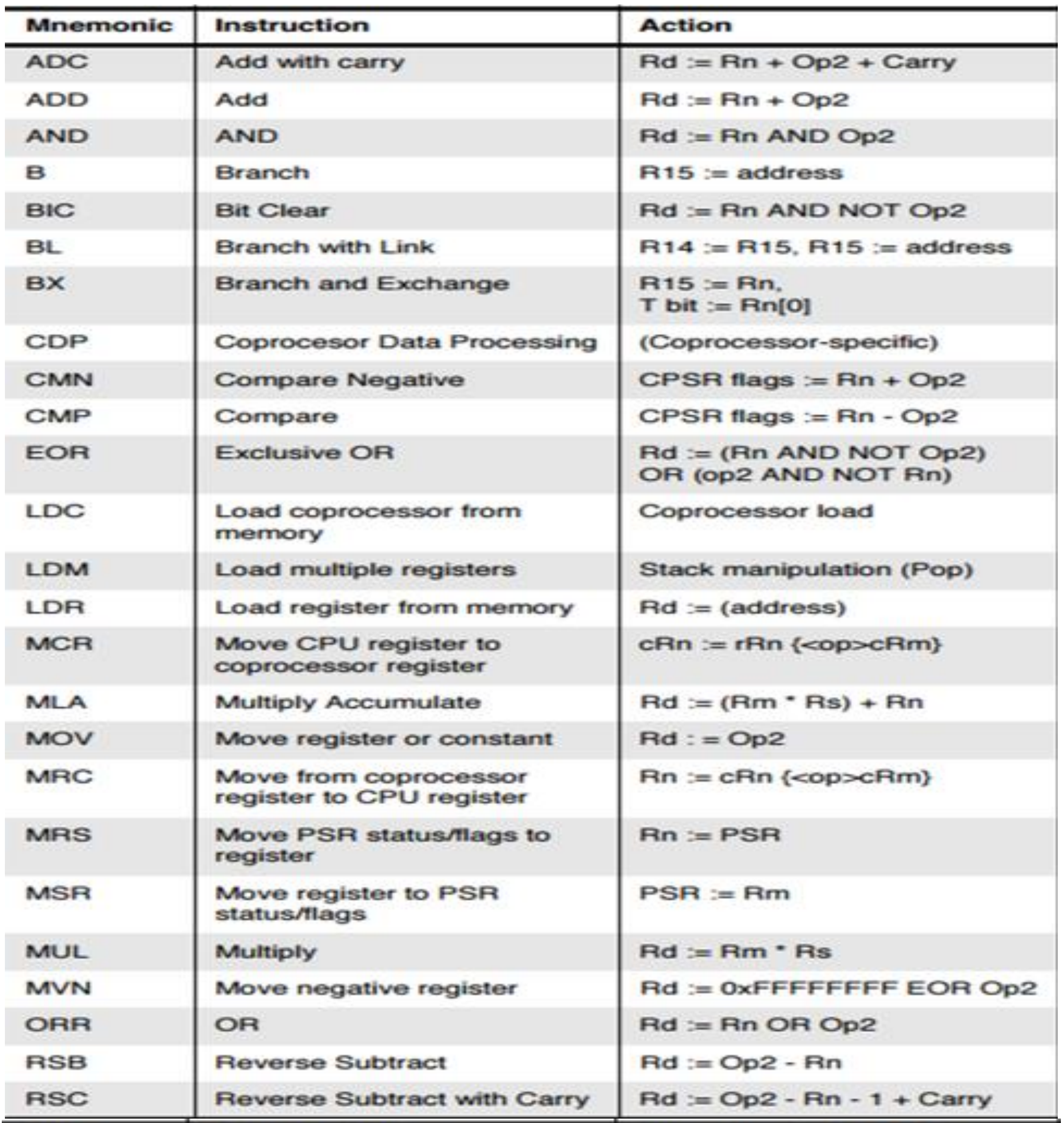
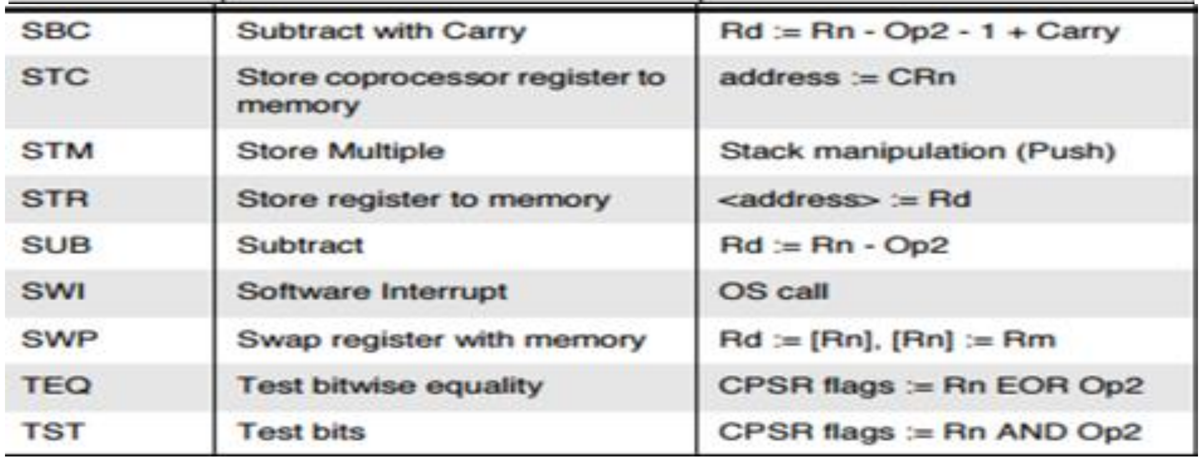
(f) Suppose that the page of the logical address 0x000100B8 got mapped into physical page number 0xFF, what is the physical address corresponding to this logical address?

Physical address is 0x0000FF0B8.

(g) 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

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