The University of Texas at Austin
Department of Electrical and Computer Engineering

# Introduction to Embedded Systems

**EE319K (Gerstlauer), Spring 2013**

---

## Midterm 1 Solutions

**Date:** February 21, 2013

UT EID: _____

Printed Name: _____
Last, First

Your signature is your promise that you have not cheated and will not cheat on this exam, nor will you help others to cheat on this exam:

Signature: _____

**Instructions:**

- Closed book and closed notes.
- No calculators or any electronic devices (turn cell phones off).
- Please be sure that your answers to all questions (and all supporting work that is required) are contained in the space (boxes) provided.
- *Anything outside the boxes will be ignored in grading.*
- For all questions, unless otherwise stated, find the most efficient (time, resources) solution.

| | | |
|---|---|---|
| **Problem 1** | 10 | |
| **Problem 2** | 10 | |
| **Problem 3** | 15 | |
| **Problem 4** | 20 | |
| **Problem 5** | 30 | |
| **Problem 6** | 15 | |
| **Total** | 100 | |

**Problem 1 (10 points): Numbers**

(a) (5 points) How many bits are minimally needed to represent all days in a year? What C data type should be used to store such values?

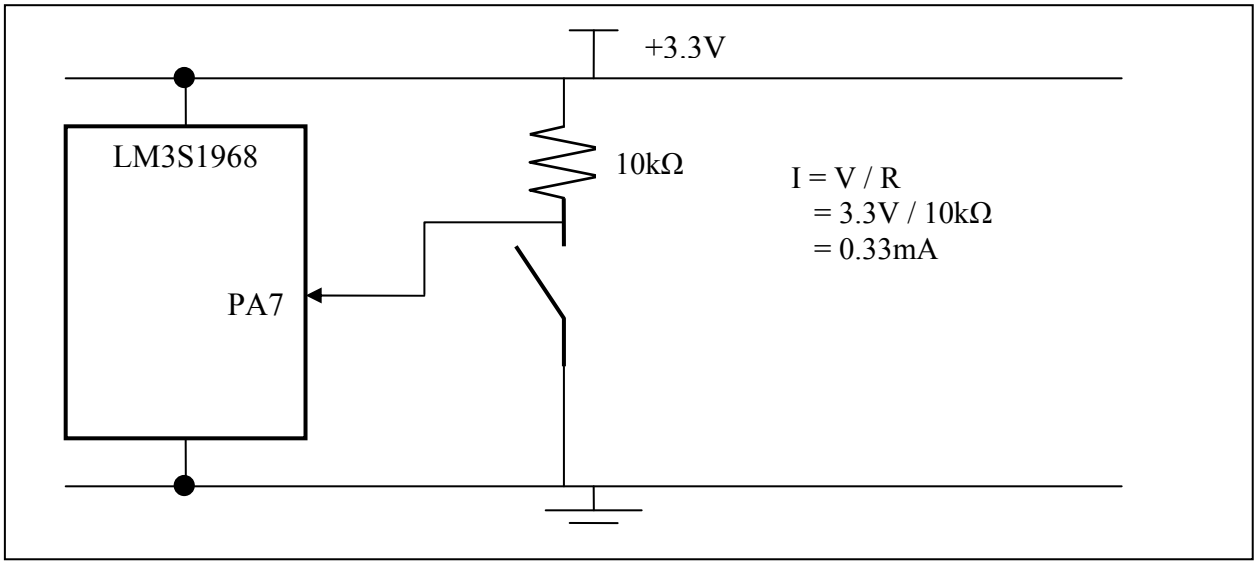
| | |
|---|---|
| Number of Bits | 9 bits |
| C Data Type | uint16_t or unsigned short |

(b) (5 points) What values has the 8-bit number 0x70 when converted to decimal and binary representations?

| Signed decimal | Unsigned decimal | Binary |
|---|---|---|
| 7 * 16 = 112 | 7 * 16 = 112 | %01110000 |

**Problem 2 (10 points): Interfacing**

Interface a switch to (input) port PA7 of the LM3S1968 using negative logic. Assuming that no current can flow in or out of the LM3S1968 and that the switch is perfect (zero resistance when closed), what current will flow through the switch when it is closed?



**Problem 3 (15 points): Arithmetic and Addressing**

(a) (5 points) For the following operation sequence, what will be the value of register R0 and condition code bits N, Z, V and C after execution of the sequence. Assume all values and registers are 8-bit wide:

| 8-bit sequence | R0 | N | Z | V | C |
|---|---|---|---|---|---|
| R1 ← -111<br>R2 ← 221<br>R0 ← R1 + R2 | 110 | 0 | 0 | 1 | 1 |

(b) (5 points) Consider the following operation sequence (in regular 32-bit ARM assembly):

```
LDR   R1,=-168
ASRS  R2,R1,#2
CMP   R1,R2
```

Mark which of the following branches will be taken after executing the above sequence:

| Branch | Taken | Not taken |
|---|---|---|
| BEQ | | X |
| BHS | | X |
| BGE | | X |
| BLO | X | |
| BLT | X | |

(c) (5 points) Consider the following assembly program:

```
      AREA CODE  ;assume this starts at address 0x0000.1000
num   DCD   0x87654321
Start LDR   R0,=num
      LDRSH R1,[R0]
```

What is the value in register R1 at the end of execution?

Big: 0xFFFF8765
Little: 0x00004321

(the ARM can be configured to different endianess and the result depends on that; default is little)

**Problem 4 (20 points): Execution**

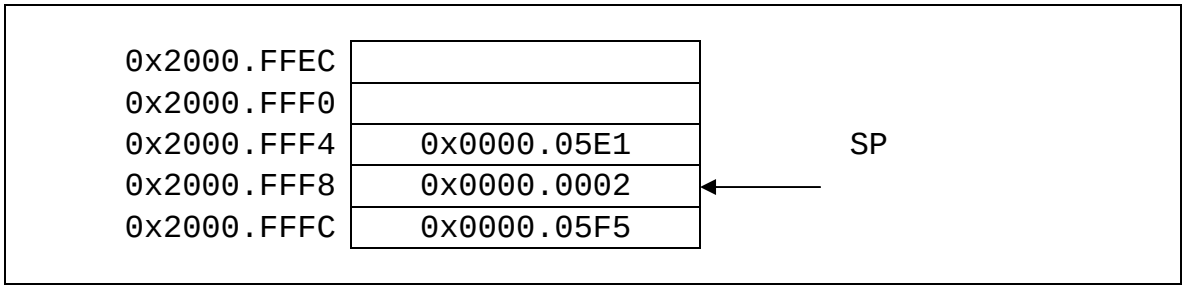
Given the following ARM assembly program:

```
                                AREA DATA
0x20000000 00000000    res      DCD  0

                                AREA CODE
0x000005D0 B500        f        PUSH     {LR}
0x000005D2 2801                 CMP      R0,#0x01
0x000005D4 D007                 BEQ      done
0x000005D6 B401                 PUSH     {R0}
0x000005D8 F1A00001             SUB      R0, R0, #1
0x000005DC F7FFFFF8             BL       f
0x000005E0 BC02                 POP      {R1}
0x000005E2 FB00F001             MUL      R0, R0, R1
0x000005E6 F85DEB04    done     POP      {LR}
0x000005EA 4770                 BX       LR

0x000005EC F04F0002    Start    MOV      R0,#2
0x000005F0 F7FFFFEE             BL       f
0x000005F4 4900                 LDR      R1,=res
0x000005F6 6008                 STR      R0,[R1]
```

(a) (10 points) Assume the stack pointer SP is initialized to 0x2001.0000. Show the contents of the stack and indicate the location of the stack pointer right after the point when the statement at address 'f' has just been executed for the second time.




(b) (5 points) What is the value in memory location 'res' at the end of execution?

2

(c) (5 points) What general functionality does the subroutine 'f' implement?

Factorial (n!)

**Problem 5 (30 points): Input/Output**

You are asked to develop a software module to control the seatbelt warning lamp as part of a car dashboard. For the part of the system that you are responsible for, the following inputs and outputs are relevant (all positive logic):

- Ports PB5…PB2 are connected to a RPM sensor that reports the current engine speed as a scaled (in units/increments of 500 RPM) unsigned 4-bit integer value, i.e. if the sensor reports a value of 2 on PB5…PB2, the engine speed is 1000 RPM.
- Port PB0 is connected to the seatbelt switch that indicates whether the seatbelt is fastened.
- Port PB7 is connected to the safety warning indicator LED.

Your subsystem is supposed to turn on the LED if the engine is running (RPM >= 1000) and the seatbelt is not fastened.

Since your code is part of a bigger system, make sure to develop subroutines that are friendly, i.e. that do not modify unrelated bits of ports. You can assume that relevant definitions are given:

```
GPIO_PORTB_DATA_R
GPIO_PORTB_DIR_R
GPIO_PORTB_AFSEL_R
GPIO_PORTB_DEN_R
SYSCTL_RCGCGPIO_R
SYSCTL_RCGCGPIO_GPIOB  (= 0x00000002, port B clock gating control)
```

(a) (10 points) Write the assembly code for the initialization subroutine of the *Belt* module. The *Belt_Init* subroutine should make PB7 an output, and PB0 and PB5 through PB2 inputs. Fill in the blanks in the code template below. You are not allowed to use bit-specific addressing or the `BIC` instruction.

```
Belt_Init
    LDR R1, =SYSCTL_RCGCGPIO_R
    LDR R0, [R1]
    ORR R0,R0,#SYSCTRL_RCGCGPIO_GPIOB
    STR R0, [R1]
    NOP
    NOP
    LDR R1, =GPIO_PORTB_DIR_R
    LDR R0, [R1]
    ORR R0,R0,#0x80
    AND R0,R0,#0xC2__
    STR R0, [R1]
    LDR R1, =GPIO_PORTB_AFSEL_R
    LDR R0, [R1]
    AND R0,R0,#0x42__
    STR R0, [R1]
    LDR R1, =GPIO_PORTB_DEN_R
    LDR R0, [R1]
    ORR R0,R0,#0xBD__
    STR R0, [R1]
    BX  LR
```

(b) (20 points) Write a main C program that first calls the *Belt_Init* subroutine from (a) then performs a loop over and over to turn the LED on iff (if and only if)

- the engine is running (RPM >= 1000), and
- the seatbelt is not fastened.

In all other cases, the LED should be off.

```
// declaration of function implemented in assembly
void Belt_Init(void);


// main program
void main(void)
{
  Belt_Init();


  while(1) {
    if((((GPIO_PORTB_DATA_R & 0x3C) >> 2) >= 2) &&
        ((GPIO_PORTB_DATA_R & 0x01) == 0)) {
      GPIO_PORTB_DATA_R |= 0x80;
    } else {
      GPIO_PORTB_DATA_R &= 0x7F;
    }
  }
}


// alternate main program
void main(void)
{
  Belt_Init();


  while(1) {
    if((((GPIO_PORTB_DATA_R >> 2) && 0x0F) >= 2) &&
        ((GPIO_PORTB_DATA_R & 0x01) == 0)) {
      GPIO_PORTB_DATA_R |= 0x80;
    } else {
      GPIO_PORTB_DATA_R &= 0x7F;
    }
  }
}
```

**Problem 6 (15 points): C Programming and Parameter Passing**

Given below is the C code for a function that checks whether a number is prime. Translate the C code into assembly. Follow the AAPCS calling convention standard, i.e. use register R0 both to pass value 'v' and return the result, and you can freely use registers R0 through R3. Note: in ARM assembly, the modulo operation (A % B) has to be implemented as (A – (B * (A / B))).

C code

```
int32_t prime(int32_t v)
{
  unsigned int32_t;

  for(i = 2; i < v; i++) {
    if((v % i) == 0) {
      return 0;
    }
  }

  return 1;
}
```

Assembly code

```
prime
      MOV  R1,#2
loop  CMP  R1,R0
      BLO  done1
      UDIV R2,R0,R1
      MUL  R2,R2,R1
      CMP  R0,R2
      BEQ  done0
      ADD  R1,R1,#1
      B    loop
done0 MOV  R0,#0
      B    done
done1 MOV  R0,#1
done  BX   LR
```

# ASCII Table

BITS 4 to 6

| BITS 0 to 3 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| | 0 | NUL | DLE | SP | 0 | @ | P | ` | p |
| B | 1 | SOH | DC1 | ! | 1 | A | Q | a | q |
| I | 2 | STX | DC2 | " | 2 | B | R | b | r |
| T | 3 | ETX | DC3 | # | 3 | C | S | c | s |
| S | 4 | EOT | DC4 | $ | 4 | D | T | d | t |
| | 5 | ENQ | NAK | % | 5 | E | U | e | u |
| 0 | 6 | ACK | SYN | & | 6 | F | V | f | v |
| | 7 | BEL | ETB | ' | 7 | G | W | g | w |
| T | 8 | BS | CAN | ( | 8 | H | X | h | x |
| O | 9 | HT | EM | ) | 9 | I | Y | i | y |
| | A | LF | SUB | * | : | J | Z | j | z |
| 3 | B | VT | ESC | + | ; | K | [ | k | { |
| | C | FF | FS | , | < | L | \ | l | ; |
| | D | CR | GS | - | = | M | ] | m | } |
| | E | SO | RS | . | > | N | ^ | n | ~ |
| | F | S1 | US | / | ? | O | | o | DEL |

**Memory access instructions**

```
LDR    Rd, [Rn]        ; load 32-bit number at [Rn] to Rd
LDR    Rd, [Rn,#off]   ; load 32-bit number at [Rn+off] to Rd
LDR    Rd, =value      ; set Rd equal to any 32-bit value (PC rel)
LDRH   Rd, [Rn]        ; load unsigned 16-bit at [Rn] to Rd
LDRH   Rd, [Rn,#off]   ; load unsigned 16-bit at [Rn+off] to Rd
LDRSH  Rd, [Rn]        ; load signed 16-bit at [Rn] to Rd
LDRSH  Rd, [Rn,#off]   ; load signed 16-bit at [Rn+off] to Rd
LDRB   Rd, [Rn]        ; load unsigned 8-bit at [Rn] to Rd
LDRB   Rd, [Rn,#off]   ; load unsigned 8-bit at [Rn+off] to Rd
LDRSB  Rd, [Rn]        ; load signed 8-bit at [Rn] to Rd
LDRSB  Rd, [Rn,#off]   ; load signed 8-bit at [Rn+off] to Rd
STR    Rt, [Rn]        ; store 32-bit Rt to [Rn]
STR    Rt, [Rn,#off]   ; store 32-bit Rt to [Rn+off]
STRH   Rt, [Rn]        ; store least sig. 16-bit Rt to [Rn]
STRH   Rt, [Rn,#off]   ; store least sig. 16-bit Rt to [Rn+off]
STRB   Rt, [Rn]        ; store least sig. 8-bit Rt to [Rn]
STRB   Rt, [Rn,#off]   ; store least sig. 8-bit Rt to [Rn+off]
PUSH   {Rt}            ; push 32-bit Rt onto stack
POP    {Rd}            ; pop 32-bit number from stack into Rd
ADR    Rd, label       ; set Rd equal to the address at label
MOV{S} Rd, <op2>       ; set Rd equal to op2
MOV    Rd, #im16       ; set Rd equal to im16, im16 is 0 to 65535
MVN{S} Rd, <op2>       ; set Rd equal to -op2
```

**Branch instructions**

```
B    label   ; branch to label    Always
BEQ  label   ; branch if Z == 1   Equal
BNE  label   ; branch if Z == 0   Not equal
BCS  label   ; branch if C == 1   Higher or same, unsigned ≥
BHS  label   ; branch if C == 1   Higher or same, unsigned ≥
BCC  label   ; branch if C == 0   Lower, unsigned <
BLO  label   ; branch if C == 0   Lower, unsigned <
BMI  label   ; branch if N == 1   Negative
BPL  label   ; branch if N == 0   Positive or zero
BVS  label   ; branch if V == 1   Overflow
BVC  label   ; branch if V == 0   No overflow
BHI  label   ; branch if C==1 and Z==0  Higher, unsigned >
BLS  label   ; branch if C==0 or  Z==1  Lower or same, unsigned ≤
BGE  label   ; branch if N == V   Greater than or equal, signed ≥
BLT  label   ; branch if N != V   Less than, signed <
BGT  label   ; branch if Z==0 and N==V  Greater than, signed >
BLE  label  ; branch if Z==1 or N!=V  Less than or equal,signed ≤
BX   Rm      ; branch indirect to location specified by Rm
BL   label   ; branch to subroutine at label
BLX  Rm      ; branch to subroutine indirect specified by Rm
```

**Interrupt instructions**

```
CPSIE  I               ; enable interrupts  (I=0)
CPSID  I               ; disable interrupts (I=1)
```

**Logical instructions**

```
AND{S} {Rd,} Rn, <op2> ; Rd=Rn&op2    (op2 is 32 bits)
ORR{S} {Rd,} Rn, <op2> ; Rd=Rn|op2    (op2 is 32 bits)
EOR{S} {Rd,} Rn, <op2> ; Rd=Rn^op2    (op2 is 32 bits)
BIC{S} {Rd,} Rn, <op2> ; Rd=Rn&(~op2) (op2 is 32 bits)
ORN{S} {Rd,} Rn, <op2> ; Rd=Rn|(~op2) (op2 is 32 bits)
LSR{S} Rd, Rm, Rs     ; logical shift right Rd=Rm>>Rs  (unsigned)
LSR{S} Rd, Rm, #n     ; logical shift right Rd=Rm>>n   (unsigned)
ASR{S} Rd, Rm, Rs     ; arithmetic shift right Rd=Rm>>Rs (signed)
ASR{S} Rd, Rm, #n     ; arithmetic shift right Rd=Rm>>n  (signed)
LSL{S} Rd, Rm, Rs      ; shift left Rd=Rm<<Rs (signed, unsigned)
LSL{S} Rd, Rm, #n      ; shift left Rd=Rm<<n  (signed, unsigned)
```

**Arithmetic instructions**

```
ADD{S} {Rd,} Rn, <op2> ; Rd = Rn + op2
ADD{S} {Rd,} Rn, #im12 ; Rd = Rn + im12, im12 is 0 to 4095
SUB{S} {Rd,} Rn, <op2> ; Rd = Rn - op2
SUB{S} {Rd,} Rn, #im12 ; Rd = Rn - im12, im12 is 0 to 4095
RSB{S} {Rd,} Rn, <op2> ; Rd = op2 - Rn
RSB{S} {Rd,} Rn, #im12 ; Rd = im12 - Rn
CMP    Rn, <op2>       ; Rn - op2      sets the NZVC bits
CMN    Rn, <op2>       ; Rn - (-op2)   sets the NZVC bits
MUL{S} {Rd,} Rn, Rm    ; Rd = Rn * Rm       signed or unsigned
MLA    Rd, Rn, Rm, Ra  ; Rd = Ra + Rn*Rm    signed or unsigned
MLS    Rd, Rn, Rm, Ra  ; Rd = Ra - Rn*Rm    signed or unsigned
UDIV   {Rd,} Rn, Rm    ; Rd = Rn/Rm         unsigned
SDIV   {Rd,} Rn, Rm    ; Rd = Rn/Rm         signed
```

**Notes** `Ra Rd Rm Rn Rt represent 32-bit registers`

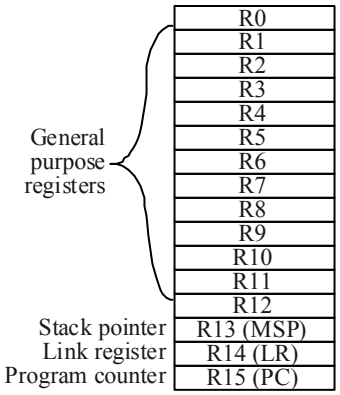
```
value   any 32-bit value: signed, unsigned, or address
{S}     if S is present, instruction will set condition codes
#im12   any value from 0 to 4095
#im16   any value from 0 to 65535
{Rd,}   if Rd is present Rd is destination, otherwise Rn
#n      any value from 0 to 31
#off    any value from -255 to 4095
label   any address within the ROM of the microcontroller
op2     the value generated by <op2>
```

Examples of flexible operand **<op2>** creating the 32-bit number. E.g., **Rd = Rn+op2**

```
ADD Rd, Rn, Rm          ; op2 = Rm
ADD Rd, Rn, Rm, LSL #n  ; op2 = Rm<<n  Rm is signed, unsigned
ADD Rd, Rn, Rm, LSR #n  ; op2 = Rm>>n  Rm is unsigned
ADD Rd, Rn, Rm, ASR #n  ; op2 = Rm>>n  Rm is signed
ADD Rd, Rn, #constant   ; op2 = constant
```

, where **X** and **Y** are hexadecimal digits:

- produced by shifting an 8-bit unsigned value left by any number of bits
- in the form **0x00XY00XY**
- in the form **0xXY00XY00**
- in the form **0xXYXYXYXY**




**Condition code bits**

N negative
Z zero
V signed overflow
C carry or
unsigned overflow