**Exam 1**, Modified for the MSPM0G3507

**Date:** Oct 4, 2017

UT EID: Professor: Valvano

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 books, no papers, no data sheets (other than the last two pages of this Exam)
* No devices other than pencil, pen, eraser (no calculators, no electronic devices), please 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/blanks will be ignored in grading*. You may use the back of the sheets for scratch work.
* You have 75 minutes, so allocate your time accordingly.
* For all questions, unless otherwise stated, find the most efficient (time, resources) solution.
* Unless otherwise stated, make all I/O accesses friendly.
* *Please read the entire exam before starting.*

|  |  |  |
| --- | --- | --- |
| **Problem 1** | 12 |  |
| **Problem 2** | 8 |  |
| **Problem 3** | 10 |  |
| **Problem 4** | 15 |  |
| **Problem 5** | 10 |  |
| **Problem 6** | 15 |  |
| **Problem 7** | 20 |  |
| **Problem 8** | 10 |  |
| **Total** | 100 |  |

**(12) Question 1.** Short answers.

**(2) Part a)** What does *VOL* mean?

**(2) Part b)** Does the equation **power = voltage\*current** apply to both resistors and LEDs? Answer yes or no.

**(2) Part c)** What does nonvolatile mean in context of computer memory?

**(2) Part d)** For what values of R0 does this code branch?

**MOVS R1,#4**

**ORRS R0,R0,R1**

**BNE FunTimes**

**(2) Part e)** Considering R0 as input and R1 as output,

what is the mathematical relationship between R1 and R0?

**LSLS R1,R0,#4**

**SUBS R1,R1,R0**

**(2) Part f)** If you add an *n*-bit signed number to an *m*-bit signed

number, what is the maximum number of bits in the sum?

Assume *n* ≥ *m*.

**(8) Question 2.** Consider the shift operation.

Why are there two shift rights, ASRS LSRS?

Why is there only one shift left, LSLS?

**(10) Question 3** Assume **Data** is an 8-bit unsigned global variable in RAM.

**uint8\_t Data;**

Write assembly code that performs the following C code (no function, just assembly code),

**if(Data >= 32){**

**Data = 255; // ceiling**

**}else{**

**Data = Data<<3;**

**}**

**(15) Question 4.** Consider the following C function with two inputs and one output.

**uint32\_t func(uint32\_t in1, uint32\_t in2){**

**uint32\_t out;**

**out = 1;**

**while(in1 >= in2){**

**out = out\*in2;**

**in2 = in2 + 1;**

**}**

**return out;**

**}**

**(5) Part a)** Assume x is a 32-bit unsigned global variable. If we were to execute **x=func(6,4);** what would be the value of x?

**(10) Part b)** Write **func** in assembly using AAPCS

**(10) Question 5.** Assume Port B pins 0, 1, 2 are inputs, and Port B pin 3 is an output. The initialization has been completed. Do not write **Init**, it is given to you. Write both C and assembly main programs that

1) Read the input pins

2) Count the number of input pins that are high.

3) If the count is 2 or 3 make the output high, if 0 or 1 make the output low

4) Pull Repeat 1-3 over and over

|  |  |
| --- | --- |
| **main:**  **BL Init**  **loop:**  **B loop**  **.end** | **int main(void){**  **Init();**  **while(1){**  **}**  **}** |

**(15) Question 6.** Assume the value of the Stack pointer (SP) is **0x20200FF8** when the following code sequence starts execution (i.e., **PC=0x00001000**). The initial memory contents in and around the SP are given on the right. When drawing the stack contents, you need only to show values on the stack that represent actual valid stack data.

Initial PC

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0x00001000 POP {R0,R1}**  **0x00001004 ADDS R2,R0,R1**  **0x00001008 BL Func B**  **0x0000100C ...**  **...**  **0x00002000 Func PUSH {R2,LR} A**  **0x00002004 MOVS R2,R1**  **0x00002008 MULS R0,R2**  **0x0000200C ADDS R0,R1**  **0x00002010 POP {R2,PC}** | |  |  | | --- | --- | | **0x20200FF4** | **1** | | **0x20200FF8** | **2** | | **0x20200FFC** | **3** | | **0x20201000** | **4** | | **0x20201004** | **5** | | **0x20201008** | **6** | | **0x2020100C** | **7** |   Initial SP |

**(6)** **Part a)** Staring with PC=0x00001000 execute until arrow A. Give the SP value and stack contents after executing of the **PUSH** instruction, as shown by arrow A:

**SP =**

|  |  |
| --- | --- |
| **0x20200FF4** |  |
| **0x20200FF8** |  |
| **0x20200FFC** |  |
| **0x20201000** |  |
| **0x20201004** |  |
| **0x20201008** |  |
| **0x2020100C** |  |

**(10) Part b)** From where you left off in part a), continue executing until you get to B, Give the SP value and stack contents while executing the instruction at memory location 0x0000100C as shown by the arrow B. Also give the values stored in R0, R1, and R2.

**SP =**

|  |  |
| --- | --- |
| **0x20200FF4** |  |
| **0x20200FF8** |  |
| **0x20200FFC** | **R0 =** |
| **0x20201000** | **R1 =** |
| **0x20201004** |  |
| **0x20201008** | **R2 =** |
| **0x2020100C** |  |

**(20) Question 7.** Assume the microcontroller’s output voltage high is 3.3V. Assume the microcontroller’s output voltage low is 0V. Do not use the 7406 driver. Pick resistors appropriately and assume you have 5V, 3.3V, and ground to which you can connect your components. The symbols for each part are given below for your convenience – *use the minimum number of parts to construct the interface*.

Part a) Interface the LED to Port B bit 7 (PB7) using negative logic. The LED operating point is 2.3V at 2mA.



Show calculations for selecting resistor value(s)

**Part b)** Interface this single pole double throw switch to the microcontroller PB6 input. The switch has two possibilities. The first case is the C pin is connected to the L pin. For this case, make PB6 low. The second case is the C pin is connected to the R pin. For this case, make PB6 high. The L pin is never connected to the R pin, and the C pin is connected to either L or R.



**(10) Question 8.** You may assume PB18 is an output. There is an array, **Times**, in ROM containing the desired set of pulse widths. The array is fixed size, but only some of the data are shown. All values in the array are T1 T2 or T4. There is a function, Delay, which takes an input in bus cycles and will wait that many cycles. Running at 80 MHz, calling Delay(80000) will wait 1 ms.

Write a function, called **wave**, that takes a parameter **t**, and

Sets PB18 high

Waits **t** bus cycles

Clears PB18 low

Waits **t** bus cycles

The prototype for this function is

Write the main program that

1) Fetches one element from the **Times** array

2) Calls **wave** with that element

3) Go to the next element and repeat steps 1 and 2

4) When you get to the end of the array, reset to beginning of array and repeat 1-2-3

Do not write the **Init** function, but do implement wave

**#define T1 80000 // 1 ms**

**#define T2 2\*T1 // 2 ms**

**#define T4 4\*T1 // 4 ms**

**const uint32\_t Times[256] = {T1, T1, T2, T4, T2, T1};**

**void wave(uint32\_t t);**

**int main(void){**

**Clock\_Init80MHz(0);**

**PB18\_Init(); // you do not need to write this**

**while(1){**

**Memory access instructions**

**LDR Rt, [Rn] // 32-bit load, EA=Rn**

**LDR Rt, [Rn,#n5] // 32-bit load, EA=Rn+n5**

**LDR Rt, [SP,#n8] // 32-bit load, EA=SP+n8**

**LDR Rt, [Rn,Rm] // 32-bit load, EA=Rn+Rm**

**LDR Rt, label2 // read contents at label2, PC rel, EA=PC+relative**

**LDR Rt, =number // Rt=number, PC relative, EA=PC+relative**

**LDRH Rt, [Rn] // 16-bit unsigned load, EA=Rn**

**LDRH Rt, [Rn,#h5] // 16-bit unsigned load, EA=Rn+h5**

**LDRH Rt, [Rn,Rm] // 16-bit unsigned load, EA=Rn+Rm**

**LDRSH Rt, [Rn,Rm] // 16-bit signed load, EA=Rn+Rm**

**LDRB Rt, [Rn] // 8-bit unsigned load, EA=Rn**

**LDRB Rt, [Rn,#imm5] // 8-bit unsigned load, EA=Rn+imm5**

**LDRB Rt, [Rn,Rm] // 8-bit unsigned load, EA=Rn+Rm**

**LDRSB Rt, [Rn,Rm] // 8-bit signed load, EA=Rn+Rm**

**STR Rt, [Rn] // 32-bit store, EA=Rn**

**STR Rt, [Rn,#n5] // 32-bit store, EA=Rn+n5**

**STR Rt, [SP,#n8] // 32-bit store, EA=SP+n8**

**STR Rt, [Rn,Rm] // 32-bit store, EA=Rn+Rm**

**STRH Rt, [Rn] // 16-bit store, EA=Rn**

**STRH Rt, [Rn,#h5] // 16-bit store, EA=Rn+h5**

**STRH Rt, [Rn,Rm] // 16-bit store, EA=Rn+Rm**

**STRB Rt, [Rn] // 8-bit store, EA=Rn**

**STRB Rt, [Rn,#imm5] // 8-bit store, EA=Rn+imm5**

**STRB Rt, [Rn,Rm] // 8-bit store, EA=Rn+Rm**

**MOV Rd2, Rm2 // move contents of Rm2 into Rd2**

**MOVS Rd, Rm // move contents of Rm into Rd, set flags**

**MOVS Rd, #imm8 // move contents of imm8 into Rd, set flags**

**MVNS Rd, Rm // set Rd equal to ~Rm (logical NOT)**

Compare and Branch instructions

**CMP Rd, #imm8 // Rd – imm8, set flags**

**CMP Rn, Rm // Rn – Rm, set flags**

**CMN Rn, Rm // Rn - (-Rm), set flags**

**B label0 // branch to label0 Always**

**BEQ label // branch if Z == 1 Equal**

**BNE label // branch if Z == 0 Not equal**

**BCS/BHS label // branch if C == 1 Higher or same, unsigned ≥**

**BCC/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 ≤**

Function call, function return, stack, and interrupt instructions

**PUSH {reglist} // push 32-bit registers onto stack, R0-R7,LR**

**POP {reglist} // pop 32-bit from stack into registers, R0-R7,PC**

**ADD Rd, SP, #n8 // Rd = SP+n8**

**ADD SP, SP, #n7 // SP = SP+n7**

**SUB SP, SP, #imm7w // SP = SP-imm7w**

**BL label1 // branch to subroutine at label1, anywhere**

**BLX Rm4 // branch to subroutine specified by Rm4, R0-R12**

**BX Rm3 // branch to location specified by Rm3, R0-R12,LR**

**CPSIE I // enable interrupts (I=0)**

**CPSID I // disable interrupts (I=1)**

**WFI // sleep and wait for interrupt**

**SVC #imm8 // software interrupt**

Logical and shift instructions

**ANDS Rdn, Rm // Rdn = Rdn&Rm**

**ORRS Rdn, Rm // Rd=Rn|Rm**

**EORS Rdn, Rm // Rd=Rn^Rm**

**BICS Rdn, Rm // Rd=Rn&(~Rm) (op2 is 32 bits)**

**LSRS Rd, Rd, Rs // logical shift right Rd=Rd>>Rs (unsigned)**

**LSRS Rd, Rm, #n // logical shift right Rd=Rm>>n (unsigned), 0 to 31**

**ASRS Rd, Rm, Rs // arithmetic shift right Rd=Rd>>Rs (signed)**

**ASRS Rd, Rm, #n // arithmetic shift right Rd=Rm>>n (signed), 1 to 32**

**LSLS Rd, Rd, Rs // shift left Rd=Rd<<Rs (signed or unsigned)**

**LSLS Rd, Rm, #n // shift left Rd=Rm<<n (signed or unsigned), 1 to 32**

*Arithmetic instructions*

**ADDS Rd, Rn, #imm3 // Rd = Rn+imm3, set flags**

**ADDS Rdn, #imm8 // Rdn = Rdn+imm8, set flags**

**ADDS Rd, Rn, Rm // Rd = Rm+Rn, set flags**

**ADD Rd2, Rm // Rd2 = Rd2+Rm**

**SUBS Rd, Rn, #imm3 // Rd = Rn-imm3, set flags**

**SUBS Rdn, #imm8 // Rdn = Rdn-imm8, set flags**

**SUBS Rd, Rn, Rm // Rd = Rm-Rn**

**RSBS Rd, Rn, #0 // Rd = 0-Rn, set flags**

**MULS Rdn, Rdn, Rm // Multiply Rdn = Rdn\*Rm, set flags**

Notes

**Rd Rdn Rm Rn Rt represent 32-bit registers R0 to R7**

**Rd2 Rm2 represent 32-bit registers R0 to R15**

**number any 32-bit value: signed, unsigned, or address**

**label0 -2048 to 2046, in multiples of 2, from PC**

**label -256 to 254, in multiples of 2, from PC**

**label2 any address within 0 to 1020, in multiples of 4, from PC**

**#h5 any value from 0 to 62 in multiples of 2**

**#n5 any value from 0 to 124 in multiples of 4**

**#n7 any value from 0 to 508 in multiples of 4**

**#n8 any value from 0 to 1020 in multiples of 4**

**#imm3 any value from 0 to 7**

**#imm5 any value from 0 to 31**

**#imm8 any value from 0 to 255**

**.data // places following lines in RAM**

**.text // places following lines in ROM**

**.align 2 // skips 0-3 bytes so the address of next line is divisible by 4**

**.equ size,10 // defines an assembly constant size with value 10**

**.byte 1,2,3 // allocates three 8-bit byte(s)**

**.short 1,2,3 // allocates three 16-bit halfwords**

**.long 1,2,3 // allocates three 32-bit words**

**.space 4 // reserves 4 bytes**

**GPIOB\_DIN31\_0** (assembly) or **GPIOB->DIN31\_0** (C code)

Read from gets the current values of the input pins of port B

**GPIOB\_DOUT31\_0** (assembly) or **GPIOB->DOUT31\_0** (C code)

Write to sets the output pins of port B; read from gets the last value written

**GPIOB\_DOUTSET31\_0** (assembly) or **GPIOB->DOUTSET31\_0** (C code)

Write 1 to bit **n** to make the output pin **n** go high

**GPIOB\_DOUTCLR31\_0** (assembly) or **GPIOB->DOUTCLR31\_0** (C code)

Write 1 to bit **n** to make the output pin **n** go low

**GPIOB\_DOUTTGL31\_0** (assembly) or **GPIOB->DOUTTGL31\_0** (C code)

Write 1 to bit **n** to toggle the output pin **n** (invert from 0 to 1 or 1 to 0)