EXAM #2

- Per MSU Academic Operating Policy 10.08, you **MUST** turn your cell phone off – not on vibrate, but completely off. If you have a pager, you must also turn it off.
- You may **NOT** use a calculator. You may use only the provided reference materials.
- When a binary result is required, give the value in hex.
- Unless stated otherwise, all multi-byte data values are stored in *little-endian* ordering.
- Answers should be clearly indicated. Placing them in a box would be ideal.
- Be as neat and well organized as possible. This is in your grade’s best interest.
- If you need additional space to work, do so on the back side of the page. Make sure it is clear where your work continues.
- Absolutely NO cheating is allowed. If you are caught in the attempt of, the act of, or the past action of academic dishonesty, you will receive the maximum punishment allowed by University policy.
- Assume all variables are near RAM. When writing code, you **must use** symbolic names for variable names, register names, and bit names for (i.e, use W1 instead of 0x0002). You do not have to show the `.space` declaration for variables. If a variable is named `u32_k`, use `u32_k` in your code, not `k`.
- Hint: A common mistake in these problems is to write code that modifies variables to the right of the `=` sign (i.e, for `a = b - c`; the code you write somehow modifies `b` or `c`, as well as `a`). This in incorrect; make sure that your code only modifies variables to the left of the `=` sign.
- Recall that `k++` is the same as `k = k + 1`, `j--` is the same as `j = j - 1`, that `i == j` is true if `i` is equal to `j`, that `i != j` is true if `i` is not equal to `j`, `<<` is a left shift, `>>` is a right shift, `|` is bitwise logical OR, `&` is a bitwise logic AND, `^` is a bitwise logical XOR. Typecasts convert one type to another: `((uint32) u8_g) + u32_h` converts `u8_g` to a `uint32` then adds that value to `u32_h`.
- Be very careful on the use of WREG vs. W0 – often, only one is correct! Refer to the instruction set summary to check every instruction you write.
- The notation `*ptr++` means first to deference the pointer, `*ptr`, then to increment the pointer, `ptr++`.

As a Mississippi State University student I will conduct myself with honor and integrity at all times. I will not lie, cheat, or steal, nor will I accept the actions of those who do.

Signature: ____________________________ Date: _____________

«Name»   «Num»
In the following problems, assume the variables used have already been declared. For example, the use of the variable \texttt{i32}_a implies an \texttt{int32 i32}_a declaration earlier in the code.

1. (20 pts) Write a PIC24 assembly code fragment to implement the following. Be sure to include the comment \textquote{; if stuff} in your code.

```assembly
if ( (i16_a < i16_b) && ( ((uint32) u8_c) + u32_d) ) {
    // If stuff - put an \textquote{; if stuff} label in your code
}

; W0
; W0 W1 | ---- W2:W1------ W4:W3
; if ( (i16_a < i16_b) && ( ((uint32) u8_c) + u32_d) )
;
; Input for i16_a < i16_b
mov i16_a, W0
mov i16_b, W1
; Process
cp W0, W1
; Output
bra LT, think_more
bra GE, end_if
think_more:
; Input for ((uint32) u8_c) + u32_d)
mov.b u8_c, WREG
mov u32_d, W2
mov u32_d + 2, W3
; Process
ze W0, W1 ; u8_c -> 32 bits
clr W2
add W1, W3, W5 ; u8_c + u32_d
addc W2, W4 W6
; Output
bra NZ, if_body
bra Z, end_if
if_body:
; if stuff
end_if:
```
2. (20 pts) Write a PIC24 assembly code fragment to implement the following. Be sure to include the comment “; Do stuff” in your code.

    do {
        // Do stuff
    } while ((i16_a >> 2) > (i16_b << 2));

    do_top:
    ; Do stuff

    ; Input
    mov i16_a, W0
    mov i16_b, W1

    ; Process
    asr W0, #2, W2
    sl W1, #2, W3
    cp W2, W3

    ; Output
    bra Z, end_do
    bra NZ, do_top

    end_do:
3. (20 pts) Write a PIC24 assembly code subroutine to implement the following C function. Recall that \texttt{pu16\_a} and \texttt{u16\_b} have already been assigned to specific registers per the subroutine calling convention. Each line of C code must be included in your solution as a comment with accompanying register assignments. Each register assignment must include a type (direct, indirect, literal), a width (8, 16, or 32 bits), and a register number.

```c
uint16 silly_assign(uint16* pu16_a, uint16 u16_b) {
    pu16_a[3] = pu16_a[u16_b];
    return pu16_a[2];
}
```

```assembly
; uint16 silly_assign(uint16* pu16_a, uint16 u16_b) 
; W0 W0 W1
silly_assign:
    ; W0 W0 W1, *2 W2
    ; pu16_a[3] = pu16_a[u16_b];
    ; --I16 W3-- ----I16 W3---
    sl W1, #1, W2 ; Convert from b elements to b bytes
    mov [W0 + W2], W3
    mov W3, [W0 + 6]; 3 elements = 6 bytes

    ; --I16 W0--
    ; return pu16_a[2]
    ; W0
    mov [W0 + 4], W0
    return
```
4. (20 pts) After the execution of ALL of the C code below, fill in the blanks below. Assume little-endian order for multi-byte values. Any variables not explicitly initialized have initial values of zero. There may be extra blanks below. Or maybe not. You decide.

```c
int16   i16_a;
int16*   pi16_b;
int32   i32_c;
int32*   pi32_d;

pi16_b  = &i16_a;   // Line 1
pi16_b[2] = 0x1234  // Line 2
pi32_d  = (int32*) &i16_a; // Line 3
*pi32_d = *pi32_d + 1; // Line 4
pi32_d++;    // Line 5
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Location</th>
<th>Contents (MUST be given in hex as 16-bit values)</th>
</tr>
</thead>
<tbody>
<tr>
<td>i16_a</td>
<td>0x0800</td>
<td>0x0001 Initial value: 0x0000 Line 4 value: 0x0001</td>
</tr>
<tr>
<td>pi16_b</td>
<td>0x0802</td>
<td>0x0800 Line 1 value: 0x0800 Line 4 value: 0x0800 (unchanged)</td>
</tr>
<tr>
<td>i32_c LSW</td>
<td>0x0804</td>
<td>0x1234 Line 2 value: 0x1234</td>
</tr>
<tr>
<td>i32_c MSW</td>
<td>0x0806</td>
<td>0x0000 Initial value: 0x0000</td>
</tr>
<tr>
<td>pi32_d</td>
<td>0x0808</td>
<td>0x0804 Line 3 value: 0x0800 Line 5 value: 0x0804</td>
</tr>
<tr>
<td></td>
<td>0x080A</td>
<td>0x0000</td>
</tr>
<tr>
<td></td>
<td>0x080C</td>
<td>0x0000</td>
</tr>
</tbody>
</table>
5. (20 points) Write a short program to count the number of pushbutton presses, given a single pushbutton switch on RB4. Toggle an LED on RA3 every second press. You must configure both pins; fill in the #define macros and short functions to properly encapsulate your design. A reminder of macro names you must use: CONFIG_Rxy_AS_DIG_INPUT(), CONFIG_Rxy_AS_DIG_OUTPUT(), ENABLE_Rxy_PULLUP(), DISABLE_Rxy_PULLUP(), ENABLE_Rxy_OPENDRAIN(), DISABLE_Rxy_OPENDRAIN().

```c
void config_pushbutton(void) {
    CONFIG_RB4_AS_DIG_INPUT();
    ENABLE_RB4_PULLUP();
    DELAY_MS(1);  // Wait for pullup to take effect
}

#define CONFIG_LED() CONFIG_RA3_AS_DIG_OUTPUT()
#define PB_PRESSED() (_RB4 == 0)
#define LED (_LATA3)

typedef enum { PRESSSED, RELEASED } STATE;

void main(void) {
    STATE e_state = RELEASED;
    uint16 u16_count = 0;
    config_pushbutton();
    CONFIG_LED();
    LED = 0;

    while (1) {
        // Your code here.
        switch (e_state) {
        case PRESSSED :
            if (!PB_PRESSED()) e_state = RELEASED;
            break;

        case RELEASED :
            if (PB_PRESSED()) {
                e_state = PRESSSED;
                u16_count++;
                if (((u16_count & 1) == 0)  // True if u16_count is even
                    LED = !LED;
            }
            break;
        }
    } // End of while (1) loop
} // End of main()
```