ECE 3724-«Section»
Microprocessors

EXAM #1

- 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.
- 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.
- 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 is 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; i == j is true if i is equal to j; i != j is true if i is not equal to j; << is a left shift; >> is a right shift; | is bitwise OR; & is a bitwise AND; ^ is a bitwise XOR; || is a logical OR; and && is a logical AND.
- 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.
- You **MUST** provide a register assignment for every problem involving translation of C to assembly.

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»
Part I.

a. (10 points) Give the machine code in HEX for the instruction \texttt{xor W1, W2, [W3]}. Show your work (in binary) by copying the bits from the datasheet and explaining your choice for each bit.

From the datasheet, the encoding for \texttt{xor Wb, Ws, Wd} is

\begin{align*}
0110 & \hphantom{1} \text{www} \hphantom{1} \text{wBqq} \hphantom{1} \text{qddd} \hphantom{1} \text{dp} \hphantom{1} \text{ps} \hphantom{1} \text{ss} \\
0110 & \hphantom{1} 1000 \hphantom{1} 1000 \hphantom{1} 1001 \hphantom{1} 1000 \hphantom{1} 0010 \hphantom{1} d = 0011 \hphantom{1} (W3); \ hphantom{1} p = 000 \hphantom{1} (Ws); \ hphantom{1} s = 0010 \hphantom{1} (W2) \\
\end{align*}

= \texttt{0x688982}.

b. (10 points) Assume that \texttt{~DOUT[2]}, the complement of third bit from the bottom of DOUT[3:0], is connected to the LOC input. Give the memory location and the value of DOUT for the following program, shown on the left; give your solution to the right by filling in ALL the blanks in the address, inst, and DOUT columns on the right.

<table>
<thead>
<tr>
<th>Address</th>
<th>Inst</th>
<th>DOUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>OUT 2</td>
<td>0010</td>
</tr>
<tr>
<td>1</td>
<td>INC</td>
<td>0011</td>
</tr>
<tr>
<td>2</td>
<td>JC 1</td>
<td>0011</td>
</tr>
<tr>
<td>3</td>
<td>JMP 0</td>
<td>0100</td>
</tr>
</tbody>
</table>

Instruction cycles: 6 
Clock cycles: 12 
Time: \(0.6\) \(\mu\)s 
(Give this in decimal, not as a fraction.)

The mov and sl instructions take 1 instruction cycle each. The bra instruction takes 2 cycles if the branch is taken and 1 if not. Summing, this code requires a total of 5 instruction cycles:

\begin{align*}
\texttt{mov} \ &- 1 \\
\texttt{sl} \ &- 1 \text{ (W0 is now 0x8000)} \\
\texttt{bra} \ &- 2 \text{ (W0 is not zero, so branch to loop)} \\
\texttt{sl} \ &- 1 \text{ (W0 is now 0x0000)} \\
\texttt{bra} \ &- 1 \text{ (W0 is zero, so branch not taken)} \\
\end{align*}

6 instruction cycles = 12 clock cycles 

\[
12 \text{ cycles} \cdot \frac{1 \text{ sec}}{20 \cdot 10^6 \text{ cycles}} = 0.6 \cdot 10^{-6} \text{ sec} = 0.6 \mu\text{s}.
\]
<table>
<thead>
<tr>
<th>Address</th>
<th>Data</th>
<th>Address</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>W0</td>
<td>0x0804</td>
<td>0x0800</td>
<td>0x1234</td>
</tr>
<tr>
<td>W1</td>
<td>0x0805</td>
<td>0x0802</td>
<td>0xFFEE</td>
</tr>
<tr>
<td>W2</td>
<td>0xFF00</td>
<td>0x0804</td>
<td>0x5678</td>
</tr>
<tr>
<td>W3</td>
<td>0x0806</td>
<td>0x0806</td>
<td>0xBade</td>
</tr>
</tbody>
</table>

Assume $Z = 0$, $C = 1$ initially.

Part II. (40 pts – 10 points per problem) Assume the above memory/register contents and initial $C$ and $Z$ values at the **START** of each instruction. After the instruction is executed, give the new values of any modified register or memory contents as well as the new $C$ and $Z$ values. When the modified item is a memory value, always give the 16-bit value, so the address should always be even. When writing a modified register value, write the entire 16-bit value. Use the name of a register, not its memory location (e.g. W0, not 0x0000). If no register is modified, write “None.”

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Modified 16-bit address/register</th>
<th>New 16-bit data value</th>
<th>new Z bit</th>
<th>new C bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>mov #0x1234, W3</td>
<td>W3</td>
<td>0x1234</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

In data transfer notation, 0x1234 -> W3.

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Modified 16-bit address/register</th>
<th>New 16-bit data value</th>
<th>new Z bit</th>
<th>new C bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>mov.b [W1], [W0]</td>
<td>0x0804</td>
<td>0x5656</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

In data transfer notation, (W1) -> (W0). Evaluating, (0x0805) -> 0x0804. After a second evaluation, 0x56 -> 0x0804, recalling that memory address 0x0805 is located at the top 8 bits of 0x0804.

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Modified 16-bit address/register</th>
<th>New 16-bit data value</th>
<th>new Z bit</th>
<th>new C bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>inc.b W2, W3</td>
<td>W3</td>
<td>0x0800</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

In data transfer notation, (W2) + 1 -> W3. Evaluating, 0xFF + 1 -> W3. The produces a carry and only modifies the lower 8 bits of W3.

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Modified 16-bit address/register</th>
<th>New 16-bit data value</th>
<th>new Z bit</th>
<th>new C bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>sl W1, #4, W2</td>
<td>W2</td>
<td>0x0805</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

In data transfer notation, (W1) << 4 -> W2. Evaluating, 0x0805 << 4 -> W2. Recall that 4 bits is one hex digit.
PART III. (30 points – 15 points per problem) Convert the following C code fragments to PIC24 assembly. You **must** give a register assignment to receive full credit for each problem; other comments are not required.

```c
uint16 u16_a, u16_b, u16_c, u16_d;
uint8 u8_i, u8_j, u8_k, u8_l;

a)  do {
    u16_a = u16_a & u16_b;
} while ( ((uint16) u8_i) < u16_a );
```

```assembly
do_top:
    ; W0 W0 W1
    ; u16_a = u16_a & u16_b;
    mov u16_a, W0    ; Input
    mov u16_b, W1
    and W0, W1, W1   ; Process
    mov W0, u16_a    ; Output
    ; } while ( ((uint16) u8_i) < u16_a );
    ; -------W2-----
    mov.b u8_i, WREG ; Input
    mov u16_a, W1
    ze W0, W2        ; Process
    cp W2, W1
    bra LTU, do_top  ; Output
    bra GEU, end_do
```
b) while ( u8_i && ((u16_a >> 6) == 0x0123) ) {
    // while body
} ;

while_top:
    mov.b u8_i, WREG ; Input / process
    bra NZ, think_more ; Output
    bra Z, end_while

think_more:
    mov u16_a, W0 ; Input
    mov #0x0123, W1
    lsr W0, #6, W2 ; Process
    cp W2, W1
    bra Z, while_body ; Output
    bra NZ, end_while

while_body:
    ; while stuff
    bra while_top

don't break
**XOR**

**Exclusive OR Wb and Ws**

<table>
<thead>
<tr>
<th>Implemented in:</th>
<th>PIC24F</th>
<th>PIC24H</th>
<th>dsPIC30F</th>
<th>dsPIC33F</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

**Syntax:**

{label:} XOR.{B} Wb, Ws, Wd

[Ws], [Wd]

[Ws++], [Wd++]

[Ws--], [Wd--]

[++Ws], [++Wd]

[--Ws], [--Wd]

**Operands:**

Wb ∈ [W0 ... W15]

Ws ∈ [W0 ... W15]

Wd ∈ [W0 ... W15]

**Operation:**

(Wb).XOR.(Ws) → Wd

**Status Affected:**

N, Z

**Encoding:**

| 0110 | 1www | wBqq | qddd | dppp | ssss |

**Description:**

Compute the logical exclusive OR operation of the contents of the source register Ws and the contents of the base register Wb, and place the result in the destination register Wd. Register direct addressing must be used for Wb. Either register direct or indirect addressing may be used for Ws and Wd.

The ‘w’ bits select the address of the base register.
The ‘B’ bit selects byte or word operation (‘0’ for word, ‘1’ for byte).
The ‘q’ bits select the destination Address mode.
The ‘d’ bits select the destination register.
The ‘p’ bits select the source Address mode.
The ‘s’ bits select the source register.

**Note:**

The extension .B in the instruction denotes a byte operation rather than a word operation. You may use a .W extension to denote a word operation, but it is not required.