Exercises 2
Memory mapped I/O, Timers, and Interrupts

Computer Organization and Components / Datorteknik och komponenter (IS1500), 9 hp
Computer Hardware Engineering / Datorteknik, grundkurs (IS1200), 7.5 hp

KTH Royal Institute of Technology
Friday 18th December, 2020

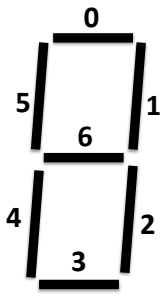
Memory Mapped I/O

1. Assume that a 7 segment display is controlled by writing to a memory mapped 32-bit I/O port located at address `0x800020f0`. The light-emitting diodes (LEDs) are organized as the figure below, where the numbers indicate the bit indices. Bit number 0 is least significant. Writing a bit value 1 means that the LED is turned on, and writing bit value 0 means that the LED is turned off.

   (a) Write down the MIPS assembly code that displays digit 3 on the 7 segment display. Use the combinations of instruction `lui` and `ori` to load the 32-bit address into a register.

   (b) Which pseudo instruction could be used instead of `lui` and `ori`? What does this mean?

   (c) Which other instruction could have been used instead of `ori`? Could the sequence of instructions be optimized and remove the need of `ori`? What is the risk of doing this?



2. Construct a C program with the following properties

   - The program should, in an infinite loop, read from a memory mapped input port at address 0x8000abc0. The returned 32-bit word indicates if a series of push buttons are pushed or not. A bit value 1 means that the button is pushed. There are 32 push buttons, each represented as a bit. The buttons are ordered in bit order, where the least significant bit represents button number 0 and the most significant bit represent button number 31. That is, the buttons are named 0 to 31, each represented with bit index 0 to 31.

- There is a memory mapped output port mapped to address 0x80007bc0. Each bit in the port represents one light-emitting diode (LED). There are in total 6 LEDs, represented by the bits 0 to 5. If the bit is 1, it means that the LED is turned on, and if it is 0, the LED is turned off.
- If button number 3 is pushed, all 6 LEDs should be turned on. If button number 3 is *not pushed*, the 6 LEDs should be turned off.

Note that your program should work, even if an optimized C compiler is used.

3. This exercise concerns PIC32 and how to use its general purpose input/output (GPIO) ports. The parallel port PORTE is controlled through the following 16 device-registers:

   - TRISE, TRISECLR, TRISESET, TRISEINV
   - PORTE, PORTECLR, PORTESET, PORTEINV
   - LATE, LATECLR, LATESET, LATEINV
   - ODCE, ODCECLR, ODCESET, ODCEINV

   TRISE is located at address 0xBF886100. The others follow, in the order specified above, with a distance of 4 bytes (since each device-register is 4 bytes wide). You can read more about these device-registers in the PIC32 Family Reference Manual. If you include the header file pic32mx.h into your C file[1], you can use the names specified above to access the device-registers of PORTE. The file includes similar definitions for all device-registers.

   (a) Write a C statement to set bits 3, 2, and 1 of PORTE as outputs, without changing the function of any other of the bits of PORTE. Use addresses to select the device-registers, not their names.

   (b) Write a C statement with the same effect as that of the previous questions, now using the name (or names) of the relevant device-registers, as defined in pic32mx.h

   (c) Explain the use of the volatile keyword in the C language.

   (d) Write a C statement that writes the bit pattern 101 to the output-bits 3, 2, and 1 of PORTE.

   (e) When the C statement from the previous question is executed, will anything be visible on the Uno32 board with Basic I/O Shield?

   (f) What will happen if a program reads the PORTE device-register?

[1] When you have installed the MCB32 tool chain, you can find this file here `/opt/mcb32/include/pic32mx.h` on Windows and Linux, and here `/Applications/mcb32tools.app/Contents/Resources/Toolchain/include/pic32mx.h` on Mac OS.

Timers

4. On a PIC32 microprocessor, the timer TMR2 is controlled through the following 14 device-registers:

   - T2CON, T2CONCLR, T2CONSET, T2CONINV
   - TMR2, TMR2CLR, TMR2SET, TMR2INV
   - PR2, PR2CLR, PR2SET, PR2INV
   - IEC0, IFS0

   You can read more about these device-registers in the PIC32 Family Reference Manual. The timer contains a 16-bit counter, which counts up at the same rate as the PIC32 processor, i.e., at an 80 MHz clock rate.

   (a) Write a C statement to set the three TCKPS bits for a 1:64 prescale value, so that the effective clock rate for the counter is reduced to (80 MHz)/64.

   (b) Write a C statement to set the Period Register for a time-out period of 10 ms, assuming that the prescale value has already been set to 1:64.

   (c) Write a C statement to reset the 16-bit counter.

   (d) Write a C statement to start the timer.

   (e) Combine the above into a complete code-sequence to initialize TMR2.

   (f) Write a C statement to test for a time-out event, i.e., whether the timer has counted all the way up to the limit set in the Period Register. The statement must contain code to mark the current time-out event as detected, so that repeating the test will not indicate a time-out event until the counter has counted all the way up to the limit once again.

Interrupts

5. The timer TMR2 can be used to interrupt the processor with regular intervals. On an interrupt, the processor will save the program counter in a special register (EPC, Exception Program Counter), disable interrupts, and then execute code at a special address. In our lab systems, the code executed upon an exception is located at the label `_isr_trampoline` in file `vectors.S`

   (a) Explain why the code at label `_isr_trampoline` saves registers $1 through $15, $24, $25, and $ra, but no others. Can that create problems in some cases?

   (b) Write a few C statements that enable the timer to interrupt the processor on a time-out event. There is a function `enable_interrupt` available; this function executes an `ei` instruction and then returns.

   (c) Write a C statement to mark the current interrupt as detected, so that there will be no timer interrupt until the counter has counted all the way up to the limit once again.