Dalhousie University
Department of Electrical and Computer Engineering

ECED 3403 – Computer Architecture

Assignment 2: An MSP 430 emulator

1 Objectives

As was shown in *Soul of a New Machine*, before a new computer system is brought to market, hardware and software designers often construct *emulators* to examine how the machine will operate under certain conditions. An emulator "imitates the function of (another system), as by modifications to hardware or software that allow the imitating system to accept the same data, execute the same programs, and achieve the same results as the imitated system".[1]

In addition to the emulation of the new computer, support software can be required if the computer has an entirely new architecture. This software can include new compilers, assemblers, linkers, and even operating systems.

In the first assignment, a two-pass assembler for the MSP 430 was developed. The second assignment is an extension of the first, requiring the design, implementation, and testing of an emulator for the MSP 430 instruction set architecture (ISA). The emulator is to load and execute S19-files produced by the assembler. A simple debugger is also required.

The emulator and assembler can run on the machine of your choice. They do not have to run on the same machine since the S19-code is transportable.

2 The Computer

The full MSP 440 ISA is to be emulated, including all CPU registers (R0/PC, R1/SP, R2/SR/CG1, R3/CG2, and R4-R15), addressing modes, instructions, and 64kiB of byte-addressable memory. External devices and interrupts are also to be supported.

2.1 Condition code register

The status register (SR/R2) is a 16-bit structure holding the current state of the CPU:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved							V	SCG1	SCG0	O-off	C-off	GIE	N	Z	C

C	Carry bit. Set when the operation (arithmetic or shift/rotate) exceeds 8-bits (byte operations) or 16-bits (word operations). The carry bit is normally used with unsigned arithmetic.
Z	Zero bit. Set when the byte or word operation (arithmetic, shift/rotate) on the target register results in a zero value; cleared when it results in a non-zero value. All 8- or 16-bits (including the most significant bit, the sign bit) are included in the test.

[1] *Emulator*. (n.d.) *American Heritage® Dictionary of the English Language, Fifth Edition*. (2011). Retrieved May 25 2015 from http://www.thefreedictionary.com/emulator

N	Negative bit. The most-significant bit (bit 7 in byte arithmetic or bit 15 in word arithmetic) is set if the result of the arithmetic or shift/rotate operation is negative; otherwise it is cleared. Arithmetic operations are assumed to be signed.
V	Overflow bit. The overflow bit is used with signed arithmetic. It is set when an arithmetic operation exceeds 7-bits (byte arithmetic) or 15-bits (word arithmetic), cleared otherwise.
GIE	General Interrupt Enable bit. Set enables maskable interrupts; cleared disables them.
SCG1	System clock generator 1. *Ignored for this assignment.*
SCG0	System clock generator 0. *Ignored for this assignment.*
O-off	Oscillator off. *Ignored for this assignment.*
C-off	CPU off. Turns off CPU when set. *See section 2.2, below.*

2.2 Interrupts and devices

The 430 has a 32-entry interrupt vector table stored in high memory, locations 0xFFC0 through 0xFFCE. Each 16-bit vector (i.e., ISR address) is associated with a priority; with 0 being the lowest (location 0xFFC0) and 31 being the highest (location 0xFFFE). Priority is only applicable when two or more interrupts occur simultaneously or two or more interrupts are pending after a RETI is executed. For an interrupt to occur, the GIE must be set. When an interrupt occurs, the PC and SR are pushed onto the stack. The GIE bit is cleared after the SR is pushed. RETI pulls the SR and PC.

The assignment is to allow up to 16 devices to be used (priorities 0 through 15). Each device is associated with a one-word port consisting of a control/status register and data register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Data (I/O)								Reserved				IE	OVF	DBA	I/O

where:

I/O	Input/output enable (bit 0). Indicate whether the device is for input (set bit) or output (clear bit). This is a control bit.
DBA	Data or buffer available (bit 1). If the device is programmed to be an input device and this bit is set, it means that data is available for reading. If the device is programmed to be an output device and this bit is clear, it means that the buffer is available for writing. The bit is cleared by the device when an input byte is read and set when the device writes a byte to the buffer. The bit is cleared by the device when an output byte is written and set by the device when the byte is transmitted. This is a status bit.
OVF	Overflow (bit 2). This bit is set if the device is enabled for input and a byte is overwritten by a subsequent byte (i.e., it was not read quickly enough) or the device is enabled and the CPU writes bytes to the buffer before the device has time to transmit the byte. This is a status bit.

IE	Interrupt enable (bit 3). Indicates whether this device is enabled for interrupts. Enable by setting the bit. If the bit is not enabled, the device can still be active; however, in order to determine whether a status change has occurred, it is necessary to poll the device. This is a control bit.
Reserved	These bits are reserved for future use or device-specific encodings (bits 4 through 7).
Data (I/O)	These bits are for input or output, depending on how the device has been defined and subsequently enabled (bits 8 through 15).

The device ports are to be located in addresses 0x0000 through 0x001E. Each device is associated with a device number, which can be used to access either the device's port or its interrupt vector. For example, device 0 has port 0x0000 and vector 0xFFC0.

Device ports can be accessed as byte-pairs, the even-numbered byte being the control/status register and the odd-numbered byte being the data register.

Interrupts occur after an instruction has completed execution.

The CPU can be put into a sleep state by setting the C-off bit. This allows the machine to wait for an interrupt, thereby avoid polling and wasting power. When an interrupt occurs, the C-off bit is cleared, the GIE bit is cleared, and the PC is assigned the address of the interrupting device. Before setting the C-off bit, the GIE bit must be set, otherwise interrupts will be ignored, essentially freezing the machine. Note that when the C-off bit is set, the CPU pushes the PC and SR onto the stack and then sleeps. When the interrupt occurs, the C-bit is set.

For additional details, see section 2.6, below. More information on the 430's interrupts can be found in the MSP430x2xx Family User's Guide (available from the course website).

2.3 Instruction cycles

Each instruction and its associated addressing mode has a set number of instruction cycles. A clock should be maintained keeping track of the number of instruction cycles a program has used during its execution. The clock information can be used when emulating interrupts (see below).

The instructions and the number of cycles associated with each are available on the website.

2.4 Bus and memory

There should be separate subroutines for the CPU and bus (for accessing memory).

The CPU subroutine is to emulate the machine's CPU; this will include code to fetch, decode, and execute instructions.

The bus subroutine should take four arguments, the contents of the memory address register, the address of the memory buffer register (to allow data to be returned), the value of the control register (read or write), and an indication whether a byte (1) or a word (0) is to be accessed (by rights, this should be part of the control register); for example:

```
/* To read: */
mar = effective_address;  /* Address to be read */
```

bus(mar, &mbr, READ, byte_word);
/* mbr has contents of location specified by ‘address’ */

/* To write: */
mar = effective_address; /* Address to be written */
mbr = data;
bus(mar, &mbr, WRITE, byte_word);

The memory will require special code (i.e., soft “circuitry”) that recognize reads and writes to the port memory.

2.5 Loader

It will be necessary to include a loader as part of the emulator that can take MSP 430 machine code stored in an S19 file and load it into the memory locations specified. The program counter must be initialized to a known value so that execution can begin after a program is loaded. This information can be supplied in the S19 file in an S9 record or using the debugger (see below).

2.6 Testing interrupts

Interrupts can occur at random intervals during a program’s execution; however, for testing purposes, it makes more sense to control when the interrupt occurs. There are a number of ways in which this can be done; the following is a description of one such method.

Causing an interrupt on an interrupt-enabled input port can be done by creating a file which specifies the time of the interrupt, port, and data associated with the interrupt. When the system clock-time is equal-to or greater-than the specified interrupt time, the emulator can trigger the interrupt (this includes setting the port’s bit in the PSR and putting the data in the port’s memory location). By specifying several interrupts at the same time in the file, concurrent interrupts can be tested.

When an interrupt-enabled output port is written to, the time, port, and contents can be written to a file. Since the output is not instantaneous, a port-specific counter can be assigned a value which is decremented after each clock-tick; when it reaches zero, the port can interrupt the CPU.

Note that if the CPU is asleep, it will be necessary to update the system clock to the time of the next interrupt to allow the next interrupt to occur, otherwise the emulator will crease execution.

2.7 Debugger

A debugger is also to be part of the emulator. This is to allow the user to load programs, start a program, stop it, and inspect or change CPU registers and memory. Note that since the machine has no way of halting, you will need to introduce an instruction “HALT” that causes the machine to stop and return control to the debugger. To catch run-away programs, a control-C signal catcher should be implemented.

3 Marking

The assignment will be marked using the following marking scheme:

Design
The design description must include a brief introduction as to the purpose of the software and describe the algorithms and data structures used.

Total points: 8.

Software
A fully commented, indented, magic-numberless, tidy piece of software that meets the requirements described above and follows the design description.

Total points: 12.

Testing
A set of tests demonstrating that the software operates according to the design description. Some of the tests should exercise the software. The tests must include the name of the test, its purpose or objective, the test configuration, and the test results.

Total points: 5.

The assignment must be submitted on paper. The executable must be sent to Dr. Hughes when the assignment is submitted.

The emulator must be demonstrated before the assignment will be marked.

4 Important Dates

Available: 25 May 2017

Due: 20 June 2017 (in lab)

Demonstrations: 20 June 2017 (in lab)

Late assignments will be penalized one point per week or fraction thereof.

5 Miscellaneous

Do not discard this work when completed, as it will be used in the final two assignments.

If you are having *any* difficulty with this assignment or the course, *please* contact Dr. Hughes as soon as possible.

This is a non-trivial assignment. It should be started as soon as it is made available.

Assignments that are found to be copies of work done by either existing or former students will result in an immediate dismissal from this course.