538 Lecture Notes Week 1

Announcements

- No labs this week. Labs begin the week of September 11, 2017.
- My email: **`kclowes@ryerson.ca`**
- Counselling hours:
  - Tuesdays ENG-449 10am – 1pm

- My Lecture Notes:
  - My lecture notes will be available in DRAFT form before each lecture at D2L and at www.ee.ryerson.ca/~courses/coe538/kclowes and at the course home page.

About the course

This course is mainly about *small embedded systems*.

- By embedded, we mean a microprocessor-based system designed to accomplish a (perhaps complex) task.
- An embedded computer system differs from a *general purpose computer* (such as a PC or Mac): you can't just decide to run whatever program you want whenever you want.
- A *small* embedded system (such as a microwave oven controller that responds to user inputs from buttons, displays status and controls the cavity magnetron that generates the microwaves) is the main type of system covered in this course.
- Small embedded systems usually do not have an operating system. The software is usually written in C (with some *assembly language*) without the convenience of OS features. The programmer has to understand the hardware to determine if a button has been pushed or how to turn on some indicator light.

Some embedded systems (such as a nuclear reactor controller or the collision avoidance system on commercial aircraft) are beyond the scope of this course.

A modern automobile includes many embedded systems. Their complexity is summarized in the following exert from a New York Times article:

> "If you look at all the code in this car," Dr. Patel said, "it's easily as much as a smartphone if not more."
>
> New highend cars are among the most sophisticated machines on the planet, containing 100 million or more lines of code. Compare that with about 60 million lines of code in all of Facebook or 50 million in the Large

Hadron Collider.

(The curious can read the whole article: http://www.nytimes.com/2015/09/27/business/complex-car-software-becomes-the-weak-spot-under-the-hood.html?_r=0)

Memory

All computer systems need memory for data and instructions.

- Each memory location has:
  - 8 bits of **data** (a *byte*).
  - An **address**.
- So there are **two** numbers associated with each memory location: address and contents.
- Addresses are commonly 16 bits (the hcs12 microprocessor used in the lab), 32 bits (older PCs, iPhones/iPads/iPods, Android phones/tablets, Blackberrys, Windows 10 phones, etc), 64 bits (newer PCs).
- Addresses are almost always described as hexadecimal numbers.
- For example, we say `*0x4001 = 0xfe`, or "the contents of address 4001 (hex) is the byte `fe` (hex)".
- A *memory dump* is commonly used to display the contents of a block of memory. For example, the 32 bytes in the memory block `0x4000–0x401f` is described with the following memory dump:

```
4000 12 fe 01 ff fc 31 62 43 02 01 11 22 33 44 55 66
4010 ff fd 99 aa bb cc dd ee ff 10 20 30 40 50 60 70
```

- This indicates that:
  - `*0x4000 = 0x12 = 18 (decimal)`
  - `*0x4001 = 0xfe = 254 (unsigned decimal)`
    `OR = -2 (signed decimal)`
  - `*0x4002 = 0x01`
- 16-bit integers require 2 bytes. For example:
  - `*0x4008 = 0x0201 = 513 (decimal)`
  - `*0x4010 = 0xfffd = 65533 (decimal) = -3 (signed decimal)`
- Sometimes, individual bits have specific meanings. For example, suppose the most significant bit of location 0x4001 indicates whether an LED is on or off and the least significant bit indicates whether a switch is pressed ("1") or not pressed ("0"). Then the contents 0xfe tell us that the LED is on and the switch is not pressed.
- Other common uses for memory are:

  - Machine language instructions (which may be one or more bytes long)
  - Addresses (16-bits) pointing to other data.
  - Character codes such as ascii (7 bits) or unicode (16 bits)
  - Floating point numbers.
- It is up to the programmer to understand how the contents of any memory location is to be interpreted.

Volatile vs non-volatile memory

- Volatile memory does not retain its contents when powered off. (This kind of memory is often called RAM—Random Access Memory. (It may be either static or dynamic.)
- Non-volatile memory does retain its contents when powered off.

The Central Processing Unit (*CPU)* and Registers

- The CPU retrieves instructions from memory and executes them by performing various logic and arithmetic operations.
- It also contains specialized memory called *registers*.
- Note: The following examples relate to the hcs12 micro-controller. However, the basic principles are applicable to any microprocessor.
- For the moment we only consider 4 of these registers:
  1. The ***Program Counter*** (PC): a 16-bit register containing the address of the next instruction.
  2. ***Accumulator A***(A): An 8-bit data register used for arithmetic and logic operations and temporary data storage.
  3. ***Condition Code Register*** (CCR): Each of the 8 bits has a different meaning. For the moment, we only consider the ***Z*** bit. Instructions set this bit to "1" if they calculate a zero value; otherwise, the bit is cleared to "0".
  4. ***Stack Pointer*** (SP): A 16-bit register pointing to the "top of Stack", a last-in first-out data structure maintained by hardware. The stack grows downwards in memory. Consequently, when 8-bit (16-bit) data is pushed onto the stack, the SP is decremented by 1 (or 2 for 16-bit data). Similarly, when data is popped (taken from) the stack, the SP is incremented.

The *Fetch-Execute* Cycle

The CPU performs an infinite sequence of *Fetch* operations followed by *Execute* operations as illustrated in the diagram below:

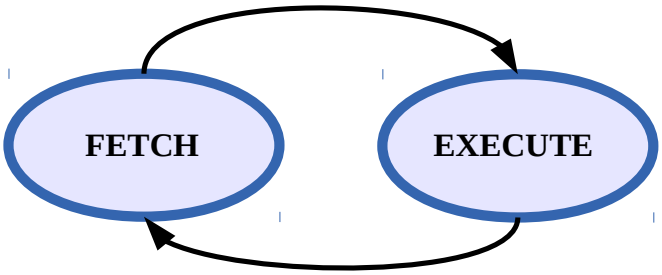


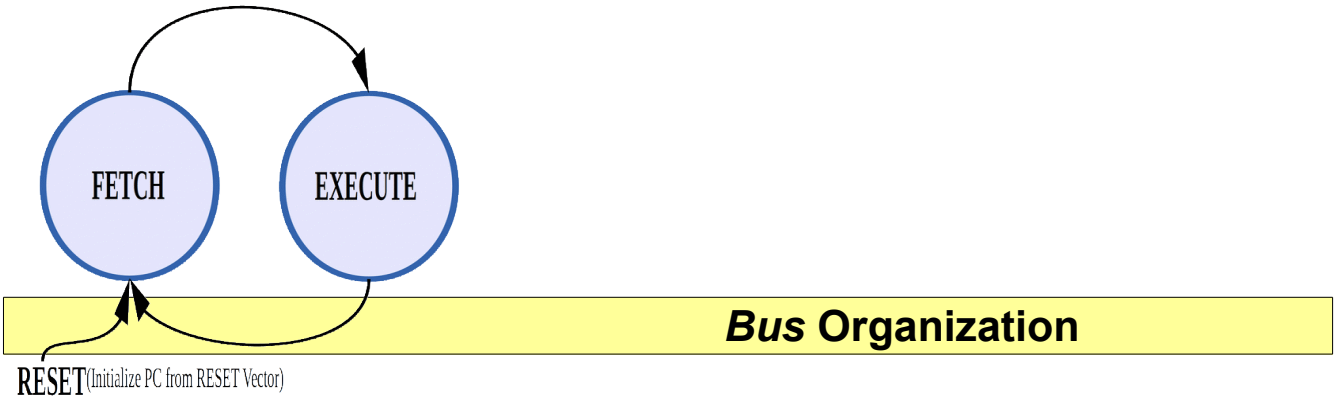
Illustration 1: The Fetch-Execute cycle

- Before starting the Fetch phase, the PC contains the address of the next machine language instruction.
- The instruction is read byte by byte during the Fetch operation. As each byte is read, the PC is incremented.
- Consequently, once the entire instruction is read, the PC is already pointing to the next sequential instruction.
- The Fetch phase is then followed by the Execute phase.
- Most instructions do not modify the PC during execution; hence the next instruction fetched/executed will be the next sequential one. However, some instructions (the *branch, jump* and return instructions) may modify the PC during the execute phase and change the normal *flow of control*. This is how conditionals, loops and *subroutines* are implemented.

Initial value of PC

- When the computer is powered on, a special signal called RESET is applied to a pin on the CPU.
- This causes the Program Counter to be initialized from a hardware defined memory location(s). This location MUST be in non-volatile memory.
- The contents of this location (the “Reset Vector”) must also be address of non-volatile memory.

• A modified version of the "Fetch-Execute" loop is shown below.



A computer system contains:

1. A CPU
2. Memory
3. Devices to communicate with the external world or do basic tasks like timing in hardware.

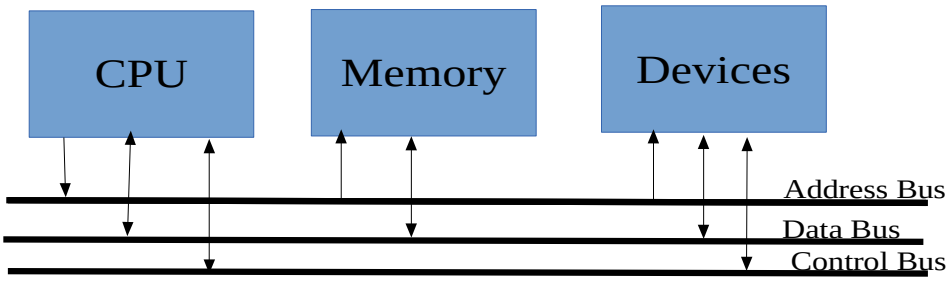
These components are interconnected with 3 *buses*:

1. The *Data Bus* over which data is transferred between components.
2. The *Address Bus* which specifies the memory address for the data being transferred.
3. The *Control Bus* which mediates the data transfer with a *clock* to control timing and functional signals to control which direction the data is moving in. Each clock period is called a *Bus Cycle.*

• When data is transferred from a memory location or device to the CPU, it is called a *Read Cycle.*

- When data is transferred from the CPU, it is a *Write Cycle*.
- If no data is transferred, it is called an *Idle Cycle.*(For example, executing a divide instruction (div) requires 10 idle cycles while the CPU performs the division.

Illustration 2: Bus organization



The bus speed (cycle time), data bus width and address bus width determine the maximum possible data transfer rate for a given computer system.

For example, given:

- 10 MHz clock (100 ns cycle time)
- 8 bit data bus (i..e 1 byte per transfer)
- 16 bit address bus.

We can say:

- The maximum transfer rate is 10 Megabytes/second
- All of memory can be read (or written) in 6.5 milliseconds.

But given:

- 2 GHz clock (0.5 ns cycle time)
- 32 bit data bus (4 bytes per transfer)
- 32 bit address bus

We can say:

- The maximum transfer rate is 8 Gigabytes/second
- All memory can be read (or written) in 0.5 seconds.

Machine language

The following table describes a small number of hcs12 machine language instructions.

Machine language	What it does	Cycles	Assembler
`86 xx`	Puts the value 'xx' into Accumulator A	`1`	`ldaa #value`
`42`	Increments AccA by 1. Sets CC Z bit to 1 if result is zero; otherwise Z bit cleared.	`1`	`inca`
`26 xx`	If Z = 0, set PC = PC + xx (where 'xx' is a signed number);otherwise, do nothing.	`3/1`	`bne target`
`20 xx`	Set PC = PC + xx (where 'xx' is a signed number)		`bra target`
`16 hh ll`	Push PC onto Stack; set PC = hhll	4	`jsr target`
`3d`	Pop top of Stack into PC	5	`rts`

Suppose memory is initialized as follows:

```
4000 86 fe 42 26 fd 20 fe
```

If the PC is initally 0x4000, what happens?

1. The first instruction (86fe) is fetched which increments the PC to 0x4002.
2. The 86fe instruction is executed, placing 0xfe into AccA.
3. The next instruction (42) is fetched incrementing the PC to 0x4003.
4. The 42 instruction is executed incrementing AccA to 0xff. Since this is not zero, the Z bit is cleared.
5. The instruction at 0x4003 (26fd) is fetched incrementing the PC to 0x4005.
6. Since the Z bit is zero, 0xfd (interpreted as the signed number -3) is added to the PC when the instruction is executed making the PC 0x4002.
7. Consequently, the next instruction fetched is 42 instruction at 0x4002 Again, fetching this instruction increments the PC to 0x4003.
8. When the instruction is executed, the 0xff value in AccA is incremented to 0x00 setting the Z bit to 1.

9. The next instruction (26fd) is fetched and the PC is incremented to 0x4005.
10. Since the Z bit is 1, executing this instruction this time does *not* change the PC. The next instruction remains at 0x4005.
11. The instruction at 0x4005 (which is 0x20fe) is fetched increasing the PC to 0x4007.
12. When executed, 0xfe (-2) is added to the PC making it 0x4005.
13. Consequently, the program enters an infinite loop where the instruction 0x20fe is fetched and executed forever.

Microcontroller vs Microprocessor vs Microcomputer etc

- Any computer system has a CPU (with internal registers), memory (both volatile and non-volatile) and peripheral devices.
- A **microprocessor** contains the CPU (and perhaps some additional circuitry) but not enough memory or peripheral devices to do anything useful without additional chips.
- A **microcontroller** (MCU) contains the CPU, memory and peripheral devices. In embedded systems it often requires little additional electronics (apart from a power supply and the devices it controls/monitors.
- The hcs12 used in this course is a microcontroller; the Intel or AMD x86 processor in your laptop is a microprocessor.
- A **microcomputer** (like a laptop or desktop general purpose computer) is based around one or more microprocessors. Each microprocessor has several "cores" where each one is a CPU.
-

Lab 1

In class discussion and demo.

Questions

1. If each pixel in an image is 24 bits (8 bits per colour) and the image (or frame) is 400 by 500 pixels, how many frames per second can be transmitted over a system where the address bus is 16 bits, the data bus is 8 bits and the bus frequency is 1 MHz? How many frames can be stored in memory. What would the answers be if the bus frequency were 2 GHz, the data bus and address bus 32 bits each?

2. Consider the following program fragment:

```
      org $4000
      lds #$5000  ;load $5000 into S register
      jsr foo
here  bra here

foo   ldaa #2
loop1:
      jsr goo
       deca
       bne loop1
       rts

goo:
     bra done
done:
      rts
```

Translate (manually) the program to machine code and show what the memory dump would look like.

Show the contents of the stack just before the instruction labelled “done” is executed.

How many times is that instruction executed before the program enters an infinite loop?

3. Consider the following memory dump:

```
4000 12 fe 01 ff fc 31 62 43 02 01 11 22 33 44 55 66
4010 ff ff 99 1a bb cc dd ee ff 10 20 30 40 50 60 70
```

If a signed 16-bit integer is stored at location 0x4010, what is its value in decimal?

An 8-bit signed integer is stored at location 0x4013. What is its value in decimal? What is its binary representation? What does its negative look like in binary and hex?

Vocabulary

Memory Dump	The standard way to display the contents of a block of memory in hexadecimal.
CPU	The Central Processor Unit, the computer's heart.
Bus	A set of parellel wires of digital signals
Address Bus	It indicates the address involved in a bus operation
Data Bus	The data being transferred during a bus cycle.
Control Bus	Signals to time and define the type of bus cycle.
IDE	“Integrated Development Environment”. Includes editors, compilers, assemblers, disassemblers, etc. We use *CodeWarrior* in this course. In

	your Java course, you used *Netbeans*. *Eclipse* is another IDE you may use in other courses.
Read Bus Cycle	Data transferred to the CPU.
Write Bus Cycle	Data transferred from the CPU to memory or a memory-mapped device.
Idle Bus Cycle	Bus inactive
Assembler	Software to translate from symbolic assembler to machine code
Disassembler	Software to translate from machine code to symbolic assembler.
Assembler directive	A line of assembler code that gives information to the assembler software and does not correspond to a machine instruction.

CodeWarrior

You can download CodeWarrior from:

http://www.freescale.com/webapp/sps/site/overview.jsp?code=CW_SPECIALEDITIONS .

Choose the CodeWarrior for HCS12(X) Microcontrollers (Classic)