538 Lecture Notes Week 4

Answers to last week's questions

1. The loop in the strcpyASM subroutine uses:

```
ldaa 1,x+
staa 1,y+
bne loopStrcpy
```

It is suggested that the "load/store" instructions could be replaced with a single instruction `movb 1,x+,1,y+` which is a legal instruction. How many bytes are required to encode the load/store instructions and how many clock cycles are required to execute them? How many bytes and clock cycles does the "movb" instruction take? Which method do you think is better? Explain. (*Warning:* This is a tricky question!)

ANSWER:

```
ldaa 1,x+   ;Machine code: A630 (2 bytes), time: 3 cycles
staa 1,y+   ;Machine code: 6A70 (2 bytes), time: 2 cycles

movb 1,x+,1,y+ ;Machine code: 180A3070 (4 bytes), time: 5 cycles
```

- Thus both the "ldaa/staa" and "movb" methods require 4 bytes of memory to encode the machine instructions and take 5 clock cycles.
- It might be argued that the "movb" is superior since it does not use AccA.
- Alas, there is a **very severe problem** with the "movb" method. It won't work!
- Why?
- Because the CCR is unaffected by a `movb` instruction and hence the conditional branch (bne) will not work!
- The only way to fix this would be to test the character moved with `tst -1,x` following the "movb" instruction.
- That would add an additional 2 bytes of machine code and 3 cycles of execution time.
- In short, the `movb 1,x+,1,y+` method is a very poor choice.

2. How would you write a subroutine `ucube8to8` (i.e. calculate the cube of an 8 bit number where the result fits into 8 bits.)

ANSWER:

- The biggest integer whose cube fits into 8 bits is 6 $(6^3 = 216 = 0xd8)$. Consequently, the table lookup method would only require a table of 7 bytes and it would run much faster than the algorithmic method. Furthermore, the complexity of the machine code for the algorithmic method would be more than 7 bytes longer than the table look-up method.

3. Write subroutines `ucube8to32` (8-bit unsigned cube to 32 bit result) and `scube8to16` (signed 8-bit cube to 16-bit result).

4. Determine the address of Port P and its Direction register. Configure the Port so that the most significant bit is an input and the other bits are output. Assume that the 7 outputs are connected to LEDs and that the input is connected to a (debounced) switch. Initially, turn off all the LEDs. Then write a loop so that whenever the switch changes the binary number displayed by the LEDs increments by 1.

5. Write a subroutine `strcmpASM` that compares strings pointed to by X and Y. The CCR should indicate the result of the comparison. If the two strings are identical, the Z bit should be set; if the string pointed to by X would appear before the one pointed to by Y, the C bit should be set; otherwise both the C and Z bits should be clear. The subroutine should have no side-effects (i.e. all registers except CCR should be unchanged when the subroutine returns.)

Parallel Ports, cont.

- The diagram below illustrates (conceptually) how a parallel port with a direction register works.
- The diagram illustrates only one bit of the port and direction registers. (Repeat it 8 times for the full diagram.)
- *Tri-state* buffers are controlled by the direction register and determine how the external pin can be "seen" by the computer's data bus.
- (Recall that a tri-state buffer connects its output to its input when activated; when unactivated, the output is disconnected from the input.)
- When a write cycle (indicated by lines on the control bus) is in progress and the address bus indicates the address of the Direction Register, a clock pulse is generated to latch the contents of the address bus into the Direction register.
- Such a cycle would occur with an instruction like `staa PortDirection`.
- If a "1" is written to a bit, the Q output is "1" which activates the tri-state buffer that connects the Q output of the Port Register bit to the external pin on the microcontroller chip.
- Writing to the port register itself with an instruction like `staa Port` will latch the contents of the accumulator into the port register and the Q outputs of the individual flip flops will appear

at the pins which the Direction register has configured as outputs.

- Note that the Direction register only has to be written to once (if the direction of the pin does not change during the application). Once configured as an output, any write to the Port register will immediately be reflected on the pin.
- When a pin is configured as an input (writing "0" to the corresponding Direction Register bit), the tri-state buffer between the Port register and the pin is de-activated and the output of the Port register is no longer reflected on the pin.
- However, when a read cycle of the address corresponding to the Port occurs (with an instruction like `ldaa Port`), the tri-state buffer between the pin and the Data Bus is activated and the external pin is effectively connected directly to the data bus during the cycle.
- Consequently, the digital value of the pin would be latched into Accumulator A at the end of the read cycle.

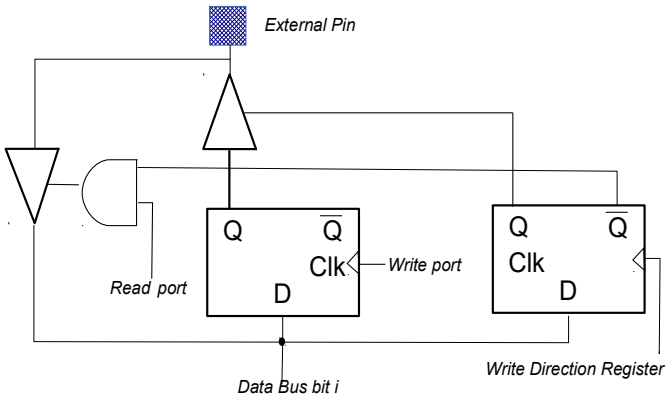


Illustration 1: One bit of a parallel port

Overview of Memory Mapped Devices

- Most devices have 3 kinds of registers:
  1. Data Registers: communicate data to/from CPU. Usually read/write.
  2. Control Registers: to configure/control the device. May be write-only.
  3. Status Registers: to monitor the device. May be read-only.
- The device connects to the Control Bus with the E clock (for timing of transfers) and the R/W line indicating whether it is a read or write cycle.
- The device has RS (Register Select) inputs. For example, if there were 16 8-bit registers, there would be 4 RS inputs.
- The RS inputs are connected to the low order bits of the address bus.
- The other address bus lines are used to select the device.
- For example, if a 16-register device is mapped to 0x9120 – 0x912f then the 12 high order bits must be 0x912 to select the device and the low order bits select one of the 16 registers (numbered 0—15).
- (Note: some registers may be 16-bits wide. Since the address bus selects bytes, each 16-bit register would occupy 2 address locations. For example, if the device had 7 16-bit registers and 2 8-bit registers, 4 RS lines would be required.)

The LCD Display

- The LCD device used in the lab is designed to connect to the CPU bus.
- However, the bus of the hcs12 is not available externally in our configuration.
- Consequently, various Port bits are used to simulate the control bus lines.
- The schematic diagram on page 6 of Lab 2 shows the connections.
- For example:
  - The R/W line is grounded. This means we can only write to this chip. (If we could read the chips status register, we could know when the LCD controller had finished processing the previous data sent to it. Since this is impossible, we have to insert a worst-case delay loop after sending data to the controller. Consequently, we cannot access the controller at maximum speed and consume CPU time in software delays.)
  - There is only a single Register Select (RS) line. So there are only 2 internal registers: 0 for data and 1 for control/status. (The most significant bit of this register indicates status, the lower 7 bits are for control.)
  - The RS pin is connected to bit 7 of Port E (PE7). Consequently, PE7 must be configured

as output. Before writing to the LCD data register, we must clear that bit; similarly, we must set it before writing to the control register.

- The E pin (clock) is connected to PE4. So that bit must also be configured for output. The programmer is responsible for generating a pulse on that bit to achieve a data transfer.
- The interface to the LCD controller is further complicated by the fact that the controller's registers are 8-bits but we only have 4 available port lines left!
- To transfer 8-bits, we need to use two 4-bit transfers. In the first transfer, we write the most significant 4 bits; the lower 4-bits are transferred in the second transfer. (See: `dataMov` subroutine on page 16 of Lab 2.)

MSB of Status Register

- It is very common for the most important status information (such as has the device completed some action) to be in the most significant bit of a status register.
- Usually, the programmer is waiting for this bit to become a '1'.
- The standard way to do this is:

```
wait:  tst status
       bpl wait
```

- This can also be done with a single instruction:

```
wait: brclr status,$80,wait
```

- Such *busy waiting* loops can often be avoided by using *interrupts*.
- An *operation complete* flag bit is set by the device. How is it cleared?
- It is very common to clear the flag by writing a '1' to it.
- In short, the device sets the bit and the programmer clears the bit to indicate that the program has acknowledged that the operation has been completed.

A/D Conversion Unit

- The ATD device on the hcs12 has the following features:
  - 8 analog channels
  - 8 or 10 bit resolution (programmable)
  - More than 100,000 measures/second possible.
- There are 8 16-bit data registers, 1 8-bit data register, 5 8-bit control registers and 2 8-bit status registers as well as reserved test registers. (The device occupies 32 bytes of memory space; i.e. it would have 5 RS lines if it were an external chip.)
- Sufficient detail on the operation is given in Lab 3.

- Complete specs are available in the MC9S12C128V1 data sheet available on the course home page.

Introduction to Interrupts

- It is very common for a "operation complete" flag to generate an *interrupt*.
- Whether an interrupt is generated is determined by how the programmer configured control registers.
- When an interupt occurs, program execution is transferred automatically to an *interrupt service routine* (ISR). Upon completion, normal program execution continues from where it was interrupted.

Questions

1. With the diagram shown for a port (single bit), what happens if the Direction Register is read? What would you add to the circuit to make this more sensible?
2. Suppose a device has 2 8-bit registers and 2 16-bit registers and that all registers cna be read and written. How many RS lines are required?
3. Write code to congure the ATD to sample all channels at (about) 1000 samples/second continuously with 8-bit resolution.
4. Show with a sequence of monitor `wb` (write byte) commands how to clear the LCD display and display the letter 'A' (ascii code: 0x41). (You need to configure PortE and PortS, generate the RS and pulse (E) signals.)
5. Consider the two ways of waiting for a status register to become "negative". How many bytes of code are required and how many CPU cycles are consumed for each loop?

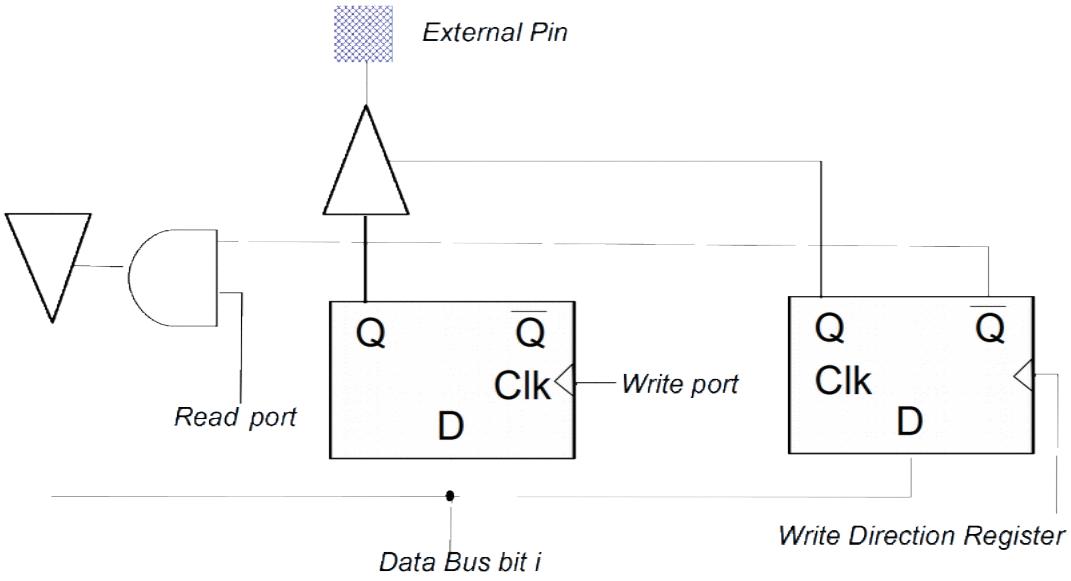
Vocabulary

New terms are in **bold**.

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.
Program Counter	A register containing the address of the next instruction.
Stack Pointer	A register containing the address of the top of stack.
Condition Code Register	Contains bits indicating various conditions (such as whether the last number added was zero, negative, generated a carry, etc.) Also called the *Status Register* in some machines (such as Intel processors).
Index Register	A register mainly used as a pointer. It's size equals the width of the Address Bus.
Arithmetic Shift	Only applies to a right shift where the sign bit is "shifted in" from the right maintaining the sign of the shifted value.
Indexed Addressing	Accessing the contents of memory whose address is calculated by adding an offset (usually zero) to an index register.
Indirect Indexed Addressing	Using indexed addressing to obtain another pointer in memory and then dereferencing the location it points to.
Overflow (V) bit	Set when the result of an addition/subtraction will not fit in the number

	of bits used.
Effective Address	The address that will be used in indexed addressing. i.e. the index register + offset.
Addressing Mode	The way an operand is specified.
Inherent Addressing	The operand is inherent in the instruction itself.
Immediate Addressing	The operand is part of the instruction (a specific field) and no further memory reference is required.
PC-Relative Addressing	The operand is an offset relative to the current value of the PC.
Subroutine	Similar to a C function. Parameters, if any, can be passed in registers or on the Stack.
Side effect	A change in the CPUs state (such as a register) unrelated to the subroutine's function.
Parameter	An argument passed to a subroutine. They are usually passed in registers or on the stack.
Direction Register	A control register associated with a parallel port that configures individual bits to be inputs or outputs.
Memory Mapped Device	A peripheral device whose internal registers are mapped to specific memory locations.

Questions


External Pin
Q
Q
Clk
D
Write port
Read port
Q
Q
Clk
D
Write Direction Register
Data Bus bit i