**Recitation #2 Topics**  
  
The main topics for this weeks recitation are:  
   Review of Homework #1  
   80x86 Instruction Set  
   General Stack Operations  
   Procedures (Arguments and Return Values)  
   Review of Homework #2 Outline  
   Conversion Functions  
   Homework #2 Testing  
   Coding Guidelines  
   Introduction to Queues  
   Lab Setup  
   Website  
  
Overheads  
   CMP instruction results  
   instruction set (class handout)  
   instructions (from 8086 book)  
   Homework Grading Criteria for Programs  
  
**Announcements**  
  
For anyone who wants to build a kit, they will have to buy the parts  
separately in the stockroom.  There is a parts list and assembly instructions  
on the web.  They should look at those instructions so they don't screw up.  
Also, we do supply them with some of the parts (mainly connectors and RAM and  
ROM).  
  
  
**Review of Homework #1**

Haven’t finished grading it yet, they’ll get it back next week

Page 1 of rec2 pdf for unsigned/signed relationships  
  
The flag conditions are arrived at by inspection.  Note that they match up  
with the conditions for

JA, JAE, JB, JBE UNSIGNED

JG, JGE, JL, JLE SIGNED

JE, JNE EQUALITY  
  
**80x86 Instruction Set**  
  
Go over the handouts and be sure everyone knows how to read them and how to  
use them.  Review the registers and addressing modes.  Make sure they  
understand the restrictions on register use and addressing modes for  
instructions (use the instruction set handout when doing this).  
  
Go over the 8086 book and point out how to read the instruction set pages  
(there are overheads of some of the pages).  Instructions to focus on  
include:  
    CMP - why it exists and when to use it  
    TEST - how it is different from CMP and when to use it  
    rotates and shifts - how they differ (use the handout)  
    CBW & CWD - how they work  
    IN & OUT - differences between ports 0-255 and 256-65535 (must use DX)  
    XLAT - how it's useful  
    Jcc - what they are for, note that many are the same  
You should probably also go thru the math instructions and go over why flags  
are important.  If there is time you may want to talk about string operations.  
  
**General Stack Operation**

CALL and RET to enter/get out of functions

NEAR CALL if function is in the same segment, FAR CALL if not

PUSH to add something to the top of the stack

POP to remove something from the top of the stack

need to balance pushes and pops or else you will mix up data

<http://wolverine.caltech.edu/eecs51/handouts/stack1.pdf>

high byte stored first on stack, removed last  
  
**Procedures (Arguments and Return Values)**  
Review how procedures work.  The three basic aspects (all discussed in  
lecture) are subroutine linkage (CALL/RET), argument passing and return  
values, and local variables.  For subroutine linkage talk about how it works  
and the difference between FAR and NEAR procedures.  For argument passing and  
return values just review the most common methods (args in registers or on  
the stack by value and return value in a register).  For local variables go  
over how to allocate space on the stack and using BP to access that space.  
You may want to go over the ENTER and LEAVE instructions.  
  
You can use either the Binary to BCD conversion procedure examples I did in  
lecture or the GCD procedure examples that are on the web site.  
  
This is very important because they need to understand this both for the  
conversion functions in Homework #2 and the queue functions in Homework #3.  
In particular for Homework #2 you should go over what a call to the function  
would look like.  Show them the assembly code for the following psuedo code:  
   Dec2String(9999, Buffer)  
  
  
  
**Review of Homework #2 Outline**

* no error handling since can only pass 16 bit
* some of you introduced special case for 0x8000, this is not necessary
* you get new argument by doing arg mod pwr10/16, not 10/16
* no return values, since written to memory
* all parts of functional spec are needed even if its none

they will be using these functions in Homework #4.  
  
Also talk about the options for output format.  For example, +1 can be output  
as 00001 or +00001 or 1 or +1 or ...  Similarly, -1 can be output as -00001  
or -1 or anything else that makes sense.  They can choose any reasonable  
format, but they need to thoroughly document that choice.  
  
**Conversion Functions**  
  
Given a template for the .asm file and they don't have to write a main loop (it is also supplied) so that should get them past a lot of the syntax issues.  
  
Show bin2bcd again

Use an index register that increments as the characters are written to write a string

The only thing that is different between this and bcd is how you write the string

They can change SI (specification does not say it has to be preserved), so  
they can just do

MOV BYTE PTR [SI], AL

INC SI.  
  
  
**Homework #2 Testing**  
  
Go over how they will have to test the Homework #2 implementation of their  
conversion functions.  They will have to assemble my test code (hw2test.asm)  
and link it to their code.  They may NOT change hw2test.asm.  In fact, when  
you demo it, you should link with a fresh copy of hw2test.obj.  My test code  
will run through the tests continuously and end up either at AllTestsGood or  
TestFailed.  They can also set breakpoints in the main code to send specific  
values into their functions.

Victor will cover in the tool tutorial at 10

look at asm86 and link86 and loc86 in tools section

asm86 hw2test.asm m1 ep db

asm86 converts.asm m1 ep db

link86 hw2test.obj,converts.obj

loc86 hw2test.lnk  
  
**Coding Guidelines**  
  
Many people use the wrong kind of loop.  I told them in lecture that if you  
find yourself initializing loop variables before entering a WHILE loop it is  
probably the wrong loop - and still many do something like:  
    r = 1  
    WHILE (r != 0)  
    ...  
Pound home the two different types of loops.  (This example probably should  
have been a REPEAT/UNTIL loop.)  
  
Duplicate code is another BIG problem.    
   r = m MOD n  
   WHILE (r != 0)  
      ...  
      r = m MOD n  
   ENDWHILE  
The calculation is r is repeated.

1. creates more work for program
2. hard to maintain

EQU constants should be used for any constant that might change or has meaning other than it’s actual value.

Don’t EQU 10 or 16

EQU ASCII\_ZERO, etc

When 0 is FALSE or <null>, it should be an EQU

Can do arithmetic on constants

Should only use an EQU for its function, don’t use ASCII\_ZERO as 0x30 because it fits

EX ASCIIZero EQU '0'

Don’t exit from procedures in the middle of loops

Thatis, a WHILE loop would have an IF in the middle of it with a RETURN.  This is  
just plain bad coding practice.  Subroutines should have only two exit points  
- at the start after any argument error checking and at the end after the  
calculation is over.  NEVER in the middle and REALLY NEVER in the middle of a  
loop.  The middle of the procedure is a problem because it is easy to miss  
and therefore can cause maintenance problems.  The middle of a loop is bad  
because it means there is effectively another exit condition for that loop  
which is not reflected in the loop conditional.  This makes the code harder  
to understand.  
  
**Introduction to Queues**

Queues are a data structure with two points head and tail

Add data to the tail pointer and remove entries at the head pointer

Can make head or tail pointer fixed and use simple array to hold queue but have to shift entries every time

Instead use a circular queue so all queue arithmetic is done modulo queue size

Normally pick power of 2 as queue size because modulo 2^n is the same as AND 2^n-1

Need to keep a hole in circular queue (place with no data) or can’t tell the difference between an empty queue and full queue

Updates to the pointers can be done pre or post increment  
  
   Head Ptr     Tail Ptr             Full                Empty  
     pre          pre            head == (tail+1)    head == tail  
     pre          post           head == tail        (head+1) == tail  
     post         pre            head == (tail+2)    head == (tail+1)  
     post         post           head == (tail+1)    head == tail  
  
Again remember that the +1 and +2 are modulo queue size.  Since these checks  
will often be done in an interrupt routine and we, ideally, want interrupt  
routines to be short we would like the checks for empty/full in the interrupt  
routines to be as fast and as simple as possible.  Ideally that means we  
would use case 1 for a transmit queue (where we worry about being empty in  
the interrupt handler) and case 2 for a receive queue (where we worry about  
the queue filling in the interrupt handler).  Since it is kind of a hassle to  
do the two queues differently you generally just pick one method (not case 3)  
and then you will use that for both transmit and receive queues and just  
accept the overhead.  
  
In recitation you can tell them how to make a queue work by giving them the  
following start of an outline.  For enqueue it would look something like:  
   wait for queue not full  
   add value to queue at the tail  
   update the tail pointer  
Don't go into any more detail than that, but tell them they need to describe  
it in more detail.  You could mention that the wait for queue not full is a  
while loop.  Also be sure to point out and explain that the order of "add  
value to queue at the tail" and "update the tail pointer" is VERY important,  
especially when interrupts are involved.  
  
Lastly, go over how a structure could be used for the queue data to make it a  
lot easier to write the code.  I talked about this some in lecture.  Don't  
give them the queue structure, but talk about what they might want to put in  
it.  
  
**Lab Setup**  
Go through the lab equipment and how to run things.  Quickly go over the  
major programs we use (tools).  At this point just do ASM86, ASM86CHK,  
LINK86, LOC86, ICE, and the serial debugger.  The editors they should be able  
to pick up on their own.  Be sure to give them the typical command lines for  
these programs.  Look at the web page for examples.  
  
You should also push ASM86CHK.  It does point out a lot of common programming  
mistakes.  They should run their program through it after it has passed the  
assembler (no assembly language errors).  
  
If you want, you can talk about editor options.  On the machines are the  
standard windows editors, emacs, Vim, and notepad++.  Emacs or notepad++  
would be my suggestion.  Also emacs has an ASM86 mode which will automatically  
indent and comment their code (see the web page for details).  
  
Finally, make it clear that absolutely NOTHING is to leave the lab.  This  
includes cables, logic probes, power supplies, etc.  And they should not move  
things around or disconnect boards.  Also point out how the problem tags work  
and the broken equipment box.  
  
Remind them that the local hard drives will be wiped at various times  
throughout the term.  So they shouldn't store things on them.  I will try to  
preserve the E drive when I update the other drives (C and D).  The U drive  
is, of course, always good.  
  
**Website**  
  
This week point out the following pages on the website:  
   Section Due Date Calendar (linked off Recitation section page)  
   ASCII Chart (References page)  
   Instruction Set (References page and Syllabus page)  
   Setting up a Development Environment (References page)  
   Getting Started with the 80x86 Tools (references page)  
   Files and the 80x86 Tools (references page)  
   Assembly Language Function Headers (references page)  
   File Headers (references page)  
   Assembly Language Explained - factor.asm (references page)  
   Homework #2 Resources