CSC201, SECTION 002, Fall 2000: Homework Assignment #3

DUE DATE

October 25 for the homework problems, in class.
October 27 for the programs, in class.

INSTRUCTIONS FOR HOMEWORK PROBLEMS

- Neat, in order, answers easy to find.
- Staple the pages together at the upper left corner.
- Fold lengthwise. On the outside write the course number, the assignment number, the date, and your name.

Thanks for your help.

PROBLEMS

1.
    1. (1) What are the advantages of having more operands (i.e, 2- or 3- address instruction format) in each instruction?
    2. (1) What are the advantages of having fewer operands (i.e., 0- or 1- address instruction format) in each instruction?

2. (4) Show how the following expression would be evaluated using Pentium instructions, assuming those instructions were a) 3-address, b) 2-address, c) 1-address, or d) 0-address. In every instruction, assume the destination must be a register, but any source (i.e., non-destination) operands can be either in memory or registers. Assume each operand and result takes one doubleword. For the 1-address case, assume the EAX register is used as the accumulator. For the 0-address case, assume the only instructions which have an operand are "push oper" and "pop oper".

var1 = var2 + (var3 * var4) - var5 / var6

[Note: In answering the questions below about addressing modes, assume that specifying a register requires 4 bits, specifying a memory address requires 32 bits, and specifying a constant requires 8 bits. You may ignore the bits which are needed to specify what addressing mode is used in the instruction.]

3.
   1. (1) Using Pentium instructions and *immediate mode* addressing, show how to subtract 5 from variable "myvar1", which is a doubleword.
   2. (1) How many bits are needed in the instruction to specify the operands?

4.
   1. (1) Using Pentium instructions and *direct mode* addressing, show how to add the contents of variable "myvar1" to register EAX.
   2. (1) How many memory (reads or writes) accesses are needed by this instruction, not counting the fetch of the instruction?

5.
   1. (1) Using Pentium instructions and *register mode* addressing, show how to move (copy) the contents of register EBX to register EAX.
   2. (1) How many bits are needed in the instruction to specify the operands?

6.
   1. (2) Using Pentium instructions and *register indirect mode* addressing, show how to add EAX to the third element of a doubleword array called "myarray" (the third element is "myarray[2]" in C).
   2. (1) How many memory (reads or writes) accesses are needed by this instruction, not counting the fetch of the instruction?

7.
   1. (2) Using Pentium instructions and *based-displacement mode* addressing, show how to copy the 9th element of a byte array called "mystring" (the 9th element is "mystring[8]" in C) into register AL. Show two alternative syntaxes for your instruction, both of which are allowable.
   2. (1) How many bits are needed in the instruction to specify the operands?

8.
   1. (2) Using Pentium instructions and *based-indexed mode* addressing, show how to move 0 into the element in the 6th row and 3nd column of a doubleword array called "myarray" (this is "myarray[5][2]" in C). Show two alternative syntaxes for your instruction, both of which are allowable.

9. Given the C language declaration:

```
struct mystruct {
        char    class;  /* 8-bit character */
        int     weight;  /* 32-bit integer */
        int     height; /* 32-bit integer  */
}       structarray[10];
```

   1. (1) Write the Pentium code to copy the weight from the 8th element of the array (i.e., (structarray[7]).weight) into EAX, using *based-indexed* addressing.
   2. (1) Write the Pentium code to set the class of the 4th element of the array to "O", using *register indirect* addressing.

10. Suppose an architecture uses the segmented memory model. In this model, a memory operand's address is specified in an instruction using *n* bits. This address is added to the contents of a segment register with *m* bits to get the operand's "effective address" in memory.
    1. (1) If $m = 24$ and $n = 16$, how many different bytes of memory can be addressed without changing the segment register value?
    2. (1) How many different bytes of memory can be addressed altogether (i.e., if both the segment register and operand address are allowed to change)?

11. (1) Suppose we want to copy the first byte of a doubleword variable "mydoubleword" into register BH. Show how to write the instruction so that it is completely clear (and explicit) that you want to copy one byte only.

12. (2) Show how you can use the Pentium stack instructions to rearrange the contents of registers EAX-EDX so that EAX has the old value of EDX, EBX has the old value of ECX, ECX has the old value of EBX, and EDX has the old value of EAX. Use only push and pop instructions, and no other registers or memory locations.

13. Suppose we have declared the stack using ".stack 1000h", and the stack is located in memory so that the address of its first byte (at the bottom of the stack) is 00402200h.
    1. (1) How many doublewords will this stack hold (give your answer in decimal)? To what is ESP initialized?
    2. (2) What is ESP after doing "push 44AA4400h", followed by "push 22000022h", followed by "0AAAA0000h"? Show the contents of the stack also.
    3. (1) What is ESP after doing 1 doubleword pop (after the above 3 pushes)?

14. (2) Show the code to check for a full stack, if the stack is 256 bytes total. Include any code needed to initialize variables, and assume all pushes and pops are of doublewords only.

15.
    1. (2) Show the Pentium code to dynamically allocate a "chunk" of 32 bytes of memory from the stack, and then to copy EAX into the 3rd doubleword (counting from the bottom, or highest-numbered byte address) of the chunk. Include any code needed to initialize variables, and use EBP as your base pointer.
    2. (1) How can you "deallocate" this chunk, and what do you have to be sure of before you do this?

16. (1) If "myvar1" and "myvar2" are two doubleword variables in memory, is the instruction "mov myvar1, myvar2" legal? Why or why not?

17. (1) Is the instruction "add 1, myvar1" legal? Why or why not?

18. (1) Is the instruction "add AH, 1" legal? Why or why not?

19. (2) We want to store the address of element "mychar[3][i]" (myvar is a byte array) into register EAX. The value of i is stored in register EBX. Show how to do this with the minimum number of instructions.

20. (2) Write the Pentium code to subtract BL from AL and determine if overflow occurred (and branch to location "printerror" if it did). BL and AL are twos-complement operands.

21. (2) Write the Pentium code to add BL to AL and determine if overflow occurred (and branch to location "printerror" if it did). BL and AL are unsigned operands.

22. (2) Besides being faster, is there any difference between "ineg EAX" and "imul EBX", where EBX has previously been initialized to hold the value -1? If so, what is it?

23. (2) Write the code to do unsigned multiplication of EBX and ECX, and branch to location "largeproduct" if the result requires more than 32 bits to store.

24. (2) Write the Pentium code to compute "EDX <-- myvar * 13", where myvar is a twos-complement doubleword.

25. (2) Show the code to divide "myvar1" by "myvar2" and put the quotient into "q1" and the remainder into "r1", where all operands are twos-complement doublewords. Include the check for overflow first and branch to "printerror2" if overflow will occur.

26. (2) Write the code to use the "rol" and "jc" (or "jnc") instructions to count the

number of bits = 1 in the doubleword variable "myvar", and leave the resulting count in ECX.

27. (2) Write the code to logically shift an operand "myvar" left by 10 positions, with the result in "myvar". Write your code as efficiently as possible.

28. (2) Write the code to compare the operands EAX and EDX, and branch to "agreatd", "aeqd", or "alessd", depending on whether EAX is greater than, equal to, or less than EDX, respectively.

29. (1) Suppose the label "alessd" above is 500000 bytes away in memory from the place where the branch occurs (this is a big program!). Show the code to conditionally branch to alessd that will work correctly in this case.

INSTRUCTIONS FOR PROGRAMMING

- Print off your program and hand in, folded lengthwise, stapled, with you name, course number, assignment number, and date on the outside.
- Submit your program electronically also.
- In your program, include the statement ".include ../pmacros.inc" (note the ".." to find the file in the parent directory). Do NOT include the sasmacros.inc file; you will not use these macros.
- Keep all your program files strictly in your EOS locker space, with no access given to other users. Do not put your files on the local machine, such as in temp space!
- Be sure your programs include a header comment indicating the assignment, date, and your name. Include a reasonable number of comments in your code as well.

Thanks for your help.

PROGRAMMING

1. (40) Write a PENTIUM program (named "hw3a.asm") that reads a sequence of records and stores them in an array. Each record consists of a 4-digit (decimal) ID number, and a value field that is a single character in the range A-Z. The records are input on a single line (the first one), separated by a single space. The records as input by the user are guaranteed to be in ascending ID number order. (All input

lines are terminated with a single LF character. End of input is indicated when you encounter two LF characters in a row.) There is guaranteed to be at least one input record, and it is guaranteed there will no records with the same ID number. After this first line, the user will then input commands, one per line. A command is one of the following:

- `D xxxx`

  This command means to delete, or remove, from your array the element whose ID number is xxxx. It is guaranteed that the value xxxx is in your array. After processing this command, your program prints

  ```
  The updated array contains ID numbers aaaa bbbb cccc ....
  ```

  , where the values of aaaa, bbbb, etc. are the ID numbers of the elements of your array, in ascending order.

- `I xxxx y`

  This command means to insert into your array, in ascending order, the element whose ID number is xxxx, and whose value is y. It is guaranteed that the ID number xxxx is not already in your array. After processing this command, your program once again prints

  ```
  The updated array contains ID numbers aaaa bbbb cccc ....
  ```

  , where the values of aaaa, bbbb, etc. are the ID numbers of the elements of your array, in ascending order.

For instance, for the input

```
1121 A 2232 F 4454 B
D 2232
I 3343
I 5565
```

The output produced by your program should be

```
1121 2232 4454
D 2232
The updated array contains ID numbers 1121 4454
I 3343
The updated array contains ID numbers 1121 3343 4454
I 5565
The updated array contains ID numbers 1121 3343 4454 5565
```

Also allow searching of your array. In addition to the above commands, there is a third command, as follows:

- `S xxxx`

  This command means to search for the element whose ID number is xxxx. There may or may not be an element with this ID number in the array. After processing this command, your program prints either

  ```
  The record with ID number xxxx has the value y
  ```

  , where y is the value entered with ID number xxx, or

  ```
  No record with ID number xxxx was found
  ```

  will be printed.

Searching may be done using any algorithm you wish, including straightforward linear search.

After the last input command has been read and processed, your program should print the message

```
The most frequent record value is y
```

, where y is the value that occurs most frequently (i.e., if there are two records with value 'A', and two records with value 'C', and three records with value 'Z', then y = 'Z'). In the event of ties, print (arbitrarily) any one of the values that is tied for most frequent.

GRADING

This homework is graded on a scale of 100 points; the points for each problem are shown above. The homework score will be weighted to contribute 5% of your course grade.

Problems will be graded according to the following interpretation:

- All right = full credit
- halfway to mostly right = half credit
- Not much or none right = no credit