

**CMPE 230 Systems Programming**  
Homework 3

**Problem 1**

The following code is A86 assembly language code. It prints the number 255 (stored at the location labelled as `number`) using hexadecimal and binary representation. The output of the program is:

FF 11111111

The program basically works by repeatedly dividing the number by the base and pushing the remainder on the stack. The contents of the stack is then popped to print the digits in correct order. Fill in the missing places in the program.

```
code segment
    mov  _____,_____          /* (a,b) */
    mov  di,2          ; iterate formatloop twice(hex and binary)
    mov  si,10h       ; divide by 16 for hex format
formatloop:
    mov  ax,[bx]
    xor  dx,dx
    push 20h         ; push blank
    mov  cx,1d       ; character counter on stack
more:
    div  si
    cmp  dl,_____          /* (c) */
    jb  _____          /* (d) */
letter:
    sub  dl,_____          /* (e) */
    add  dl,_____          /* (f) */
    jmp  enddigit
digit:
    add  dl,_____          /* (g) */
enddigit:
    push dx
    inc  cx
    xor  dx,dx
    cmp  ax,0h
    jne  more
writeloop:          ; pop and display characters
    pop  dx
    mov  ah,02h
    int  21h
    dec  cx
    jnz  writeloop
    mov  si,2d       ; divide by 2 for binary format
```

```

    dec    di            ; check to see if we will loop
    jnz    -----      /* (h) */
progfinish:
    int    20h
number:
    dw    255d
code ends

```

Note that the instruction `jb` means jump on below.

## Problem 2

Consider the following C program which generates prime numbers:

```

/*****
#include <stdio.h>

main()
{
    int prime[50] ;
    int j ;
    int k ;
    int n ;
    int quo,rem ;

P1: prime[0] = 2 ;
    n = 3 ;
    j = 0 ;
P2: j = j+1 ;
    prime[j] = n ;
P3: if (j == 49) goto P9 ;
P4: n = n + 2 ;
P5: k = 1 ;
P6: quo = n / prime[k] ;
    rem = n % prime[k] ;
    if (rem == 0) goto P4 ;
P7: if (quo <= prime[k]) goto P2 ;
P8: k = k+1 ;
    goto P6 ;
P9: for(j=0 ; j < 50 ; j++) printf("%d ",prime[j]) ;
}
*****/

```

The following x86 program generates the 50 prime numbers using the same algorithm as the above C-program. Some instructions/operands have been intentionally left blank.

code segment

```

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; PART 1: Generate primes;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
P1:  mov    bx,prime
      mov    ____,2d      ; /* (a)          */
      mov    si,3
      mov    cx,0
P2:  inc    cx
      add    bx,2
      mov    w[bx],___    ; /* (b)          */
P3:  cmp    cx,49
      -----          ; /* (c)          */
P4:  -----          ; /* (d)          */
P5:  mov    bp,prime
      add    bp,2
P6:  -----          ; /* (e)          */
      xor    dx,dx
      div   w[bp]
      -----          ; /* (f)          */
      je    P4
P7:  cmp    w[bp],___    ; /* (g)          */
      jae   P2
P8:  add    bp,___      ; /* (h)          */
      jmp   P6
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;PART 2: PRINT PRIMES;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
P9:  mov    bp,prime
      mov    si,10d
      mov    di,0
moreprimes:
      cmp    di,50
      je    progfinish
      mov    ax,[bp]
      xor    dx,dx
      push  20h
      mov    cx,1d
nonzero:
      div   si
      add   dl,"0"
      push  dx
      inc   cx
      xor   dx,dx
      cmp  ax,0h
      jne  nonzero
writeloop:
      pop   dx

```

```

        mov    ah,02h
        int    21h
        dec    cx
        jnz    writeloop
        add    bp,2
        inc    di
        jmp    moreprimes
progfinish:
        int    20h
prime:
        dw    2
code ends

```

a) Fill in the empty boxes shown in the above assembly source. (Write your answer in the comments area). b) Look at the C-source again, and write down the corresponding operand (in PART 1 of assembly source) to the following variables:

- prime
- prime[k]
- prime[j]
- N
- quo
- rem
- j

### **Problem 3**

You are given the following hypothetical machine called **Simple32**. **Simple32** computer has 32 memory locations. Each memory location is a byte. The computer has 2 general registers (R0 and R1). The machine handles integers which are in *2's complement form*. The instruction format for the computer is as follows:

opcode	register	address
(2 bits)	r-operand	a-operand
	(1 bit)	(5 bits)

The machine can be programmed with the **Simple32** Assembly Language. The instruction set for the machine is given below.

Mnemonic	Opcode	Description
HALT	0	Stop the program
ADD	1	$(r) = (r) + (a)$
LOAD	2	load, $(r) = (a)$
STOR	3	store, $(a) = (r)$

The notation  $(r)$  denotes the contents of register  $r$  and  $(a)$  specifies the contents of memory at address  $a$ . A pseudo-instruction is also available in the Simple32 Assembly Language:

CONS specifies that its operand which is a signed decimal integer is to be placed in one byte of the memory.

Additionally, any instruction can be prefixed with a label and this label can be used as a substitute for the address of the instruction.

Given the following program:

```
LOAD R0,N
ADD R0,5
ADD R0,M
STOR R0,N
HALT
M CONS -3
N CONS -2
```

- a) Assemble the above assembly program into machine code. Write down your machine code using hexadecimal numbers.
- b) Execute the machine code and show the contents of the memory when the machine hits the HALT instruction. Again use hexadecimal numbers to show the contents of the memory.

#### **Problem 4**

Write an A86 assembly language program which reads a string terminated by newline (the string can be up to 100 characters in length) and counts the number of occurrences of the numeric characters. Your program should print the computed count.