

CMPE 230 Systems Programming
Homework 4
(due Jan. 10th)

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   -----
progfinish:
    int   20h
number:
    dw    255d
code ends

```

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

Problem 2

a) Draw the layout of the memory at the point the program reaches the line marked **here**.

b) What is meant by *memory leak*? Is there any memory leak in the program?

```

#include <stdio.h>

int *z[4] ;

void B(int b) ;

void A(int x,int *a[4])
{
    a = a + x ;
    a[0] = (int *) calloc(x+1,sizeof(int)) ;
    if (x) {
        B(x) ;
    }
}

void B(int b)
{
    extern int x ;

    x = x + 1 ;
    if (b) {
        A(b-1,z) ;
    }
    printf("%d\n",x) ; /* here */
}

int x = 2 ;

main()
{

```

```
    A(x,z) ;
}
```

Problem 3

Consider the C program:

```
/******  
#include <stdio.h>  
  
char *names[] = {"Deniz","Hulya","Akkaya", "Pamela"} ;  
  
main()  
{  
    int x, y , z ;  
    int i ;  
    char *q[3] ;  
  
    x = sizeof(q) ;  
    for (i=0 ; i < 3 ; i++) {  
        *(q+i) = *(names+i) ;  
    }  
    y = x ;  
    z = y ;  
    printf("%s %s\n",*q,*(q+2)) ;  
}  
/******
```

This C program is compiled with gcc -S example.c command on the linux which produces the following gnu assembler output:

```
.file "1.c"  
.version "01.01"  
gcc2_compiled.:  
.globl names  
.data  
.align 4  
.type names,@object  
names:  
.long .LC0  
.long .LC1  
.long .LC2  
.long .LC3  
.section .rodata  
.LC3:  
.string "Pamela"  
.LC2:  
.string "Akkaya"
```

```

.LC1:
.string "Hulya"
.LC0:
.string "Deniz"
.size names,16
.LC4:
.string "%s %s\n"
.text
.align 4
.globl main
.type main,@function
main:
pushl %ebp
movl %esp,%ebp
subl $40,%esp
movl $12,-4(%ebp)
movl $0,-16(%ebp)
.p2align 4,,7
.L3:
cmpl $2,-16(%ebp)
jle .L6
jmp .L4
.p2align 4,,7
.L6:
movl -16(%ebp),%eax
movl -16(%ebp),%edx
movl names(,%edx,4),%ecx
movl %ecx,-28(%ebp,%eax,4)
.L5:
incl -16(%ebp)
jmp .L3
.p2align 4,,7
.L4:
movl -4(%ebp),%eax
movl %eax,-8(%ebp)
movl -8(%ebp),%eax
movl %eax,-12(%ebp)
addl $-4,%esp
movl -20(%ebp),%eax
pushl %eax
movl -28(%ebp),%eax
pushl %eax
pushl $.LC4
call printf
addl $16,%esp
.L2:
movl %ebp,%esp

```

```

popl %ebp
ret
.Lfe1:
.size main,.Lfe1-main
.ident "GCC: (GNU) 2.95.2 19991024 (release)"

```

Answer the following questions about the above programs:

- (a) Circle and name all the references to the C variables `x`, `y`, `z`, `i`, `q[i]`, `names[i]`, `q[0]` and `q[2]` on the assembly code.

Note that the syntax:

$$\text{disp}(\text{base}, \text{index}, \text{scale})$$

means the following:

$$[\text{base} + \text{index} * \text{scale} + \text{disp}]$$

Problem 4

The following a86 program enters a string and determines whether it a palindrome or not. Recall that a palindrome is a word that reads the same in reverse. Some examples are: `madam`, `dad`. If the entered string is palindrome, the program prints `y`, otherwise it prints `n`. Fill in the blanks in the program.

```

code segment
    mov     bp,array
    mov     bx,array
    mov     ah,____ ; /* (a) */
input:   int     21h
    cmp     al,10d ; read string until newline
    je     testpal
    mov     ____,al ; /* (b) */
    inc     bp
    jmp     input
testpal:
    dec     bp
    mov     dl,'n'
begloop: cmp     bx,bp
    jae     ispal ; it is palindrome
    mov     cl,____ /* (c) */
    cmp     cl,____ /* (d) */
    jne     notpal ; not a palindrome
    inc     ____ ; /* (e) */
    dec     bp
    ___     ____ /* (f) (g) */
ispal:   mov     ____, 'y' ; /* (h) */
notpal:  mov     ____, 02h ; /* (i) */

```



```

        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 6

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 -1
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.