

به نام خدا
کارگاه کامپیوتر
آزمایش سوم

۸۱۰۱۹۶۶۸۸

علی بهاری

۱- برنامه ای به زبان C بنویسید که ۲۰ کاراکتر بین a تا z را به صورت تصادفی به کاربر نمایش دهد و پس از این که کاربر کل کاراکترها را تایپ کرد سرعت و دقت تایپ را محاسبه کند و نمایش دهد. کد را کامپایل کرده و اجرا و تست کنید.

```
GNU nano 2.2.6 File: test01.c

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <sys/time.h>

int main(){
    char generatedString[20];
    srand(time(NULL));
    for(int i = 0 ; i < 20 ; i++){
        generatedString[i] = (rand() % (90 - 65)) + 65;
    }
    printf("Random Text: ");
    for(int i = 0 ; i < 20 ; i++){
        printf("%c" , generatedString[i]);
    }
    char user_text[20];
    struct timeval start;
    gettimeofday(&start , 0);
    printf("\nAnswer Text: ");
    for(int i = 0 ; i < 20 ; i++){
        scanf(" %c" , user_text + i);
    }
    struct timeval end;
    gettimeofday(&end , 0);

    [ Read 37 lines ]
    ^G Get Help    ^O WriteOut    ^R Read File    ^Y Prev Page    ^K Cut Text     ^C Cur Pos
    ^X Exit        ^J Justify     ^W Where Is    ^U Next Page    ^U UnCut Text  ^T To Spell
```

تصویر ۱: قسمت اول کد

```

double takenTime = (end.tv_sec - start.tv_sec)+(end.tv_usec -
start.tv_usec) * 1e-6;
printf("Time = %f s\n" , takenTime);

int correct = 0;
for(int i = 0 ; i < 20 ; i++){
    if(user_text[i] == generatedString[i]){
        correct++;
    }
}
printf("Accuracy: %d percent\n" , correct * 5);
return 0;
}

```

^G Get Help ^O WriteOut ^R Read File ^Y Prev Page ^K Cut Text ^C Cur Pos
 ^X Exit ^J Justify ^W Where Is ^U Next Page ^U UnCut Text ^T To Spell

تصویر ۲: قسمت دوم و ادامه کد

به دلیل عدم نمایش کامل و یک جا کد در QEMU کد در دو عکس جداگانه آورده شده است. دو تصویر ۱ و ۲ کل کد قسمت اول را نمایش می دهند.

```

pi@raspberrypi ~/session04/c $ gcc -std=gnu99 test01.c && ./a.out
Random Text: XFDWGXSGFNUJEUSEILT
Answer Text: asdfghdjklsa12SEILT
Time = 16.989661 s
Accuracy: 25 percent
pi@raspberrypi ~/session04/c $ _

```

تصویر ۳: نمونه اول خروجی کد قسمت اول

```
pi@raspberrypi ~/session04/c $ gcc -std=gnu99 test01.c && ./a.out
Random Text: YWILURBLKFPXGLYIJYPN
Answer Text: 1276425ndknjGLYIJYPN
Time = 17.852164 s
Accuracy: 40 percent
pi@raspberrypi ~/session04/c $
```

تصویر ۴: نمونه دوم خروجی کد قسمت اول

خروجی کد در بخش Time مدت زمان پاسخ گویی کاربر را در real time (نه زمان CPU که از clock() استفاده می کند) بر حسب ثانیه نشان می دهد. در بخش Accuracy هم درصد تطابق ورودی کاربر و رشته تصادفی نشان داده شده است.

۲- به سه طریق برنامه ای به زبان اسمبلی بنویسید که عدد ۵ را در عدد ۸ ضرب کند و نتیجه را در تابع main برگرداند.

```
GNU nano 2.2.6      File: mul01.s

.global main
.func main

main:
    mov r0 , #8
    mov r1 , #0
    ADD r1 , r1 , r0
    ADD r1 , r1 , r0
    ADD r1 , r1 , r0
    ADD r1 , r1 , r0
    ADD r1 , r1 , r0
    mov r0 , r1
    bx lr

[ Read 13 lines ]
^G Get Help  ^O WriteOut  ^R Read File  ^Y Prev Page  ^K Cut Text   ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is   ^U Next Page  ^U UnCut Text ^T To Spell
```

تصویر ۵: کد mul01

در این کد مقدار ۸ به تعداد ۵ بار با خود جمع می شود و در نهایت مقدار ۴۰ در تابع main برگردانده می شود. امکان استفاده از یک حلقه نیز وجود دارد.

```

(gdb) start
Temporary breakpoint 1 at 0x8390
Starting program: /home/pi/session04/asm/mul01

Temporary breakpoint 1, 0x00008390 in main ()
(gdb) stepi
0x00008394 in main ()
(gdb) stepi
0x00008398 in main ()
(gdb) stepi
0x0000839c in main ()
(gdb) stepi
0x000083a0 in main ()
(gdb) stepi
0x000083a4 in main ()
(gdb) stepi
0x000083a8 in main ()
(gdb) stepi
0x000083ac in main ()
(gdb) stepi
0x4005781c in __libc_start_main () from /lib/arm-linux-gnueabi/libc.so.6
(gdb) stepi
0x400722a8 in exit () from /lib/arm-linux-gnueabi/libc.so.6
(gdb) info registers r0 r1
r0          0x8      8
r1          0x28     40
(gdb)

```

تصویر ۶: نمایش مقادیر register های کد mul01 با استفاده از GDB

```

pi@raspberrypi ~/session04/asm $ as -o ./mul01.o ./mul01.s
pi@raspberrypi ~/session04/asm $ gcc -o mul01 ./mul01.o
pi@raspberrypi ~/session04/asm $ ./mul01
pi@raspberrypi ~/session04/asm $ echo $?
40
pi@raspberrypi ~/session04/asm $ _

```

تصویر ۷: خروجی کد mul01

در دو تصویر بالا کد mul01 تست شده و مقادیر register های مورد استفاده و خروجی کد نمایش داده شده است.

```
GNU nano 2.2.6 File: mul02.s

.global main
.func main

main:
    mov r1 , #5
    mov r2 , #8
    MUL r0 , r1 , r2
    bx lr

[ Read 8 lines ]
^G Get Help ^O WriteOut ^R Read File ^Y Prev Page ^K Cut Text ^C Cur Pos
^X Exit ^J Justify ^W Where Is ^U Next Page ^U UnCut Text ^T To Spell
```

تصویر ۸: کد mul02

در این کد از دستور MUL که برای ضرب کردن مقادیر دو register است استفاده شده است.

```
pi@raspberrypi ~/session04/asm $ as -o ./mul02.o ./mul02.s
pi@raspberrypi ~/session04/asm $ gcc -o mul02 ./mul02.o
pi@raspberrypi ~/session04/asm $ ./mul02
pi@raspberrypi ~/session04/asm $ echo $?
40
pi@raspberrypi ~/session04/asm $ _
```

تصویر ۹: خروجی کد mul02

```

(gdb) start
Temporary breakpoint 1 at 0x8390
Starting program: /home/pi/session04/asm/mul02

Temporary breakpoint 1, 0x00008390 in main ()
(gdb) stepi
0x00008394 in main ()
(gdb) stepi
0x00008398 in main ()
(gdb) stepi
0x0000839c in main ()
(gdb) stepi
0x4005781c in __libc_start_main () from /lib/arm-linux-gnueabi/libc.so.6
(gdb) stepi
0x400722a8 in exit () from /lib/arm-linux-gnueabi/libc.so.6
(gdb) info registers r0 r1 r2
r0          0x28      40
r1          0x5       5
r2          0x8       8
(gdb) _

```

تصویر ۱۰: نمایش مقادیر register های کد mul02 با استفاده از GDB

در دو تصویر بالا کد mul02 تست شده و مقادیر register های مورد استفاده و خروجی کد نمایش داده شده است.

```

GNU nano 2.2.6      File: mul03.s

.global main
.func main
main:
    mov r1 , #5
    mov r0 , r1 , LSL #3 /* 8 = 2 ^ 3 so shift left 5 three times */
    bx lr

[ Read 7 lines ]
^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Page ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is  ^U Next Page ^U UnCut Text ^T To Spell

```

تصویر ۱۱: کد mul03

در کد mul03 از LSL برای شیفت چپ عدد ۵ به اندازه ۳ واحد استفاده شده است. به دلیل این که ۸ برابر ۲ به توان ۳ است پس برای ضرب عدد ۵ در ۸ کافی است ۵ را ۳ واحد به چپ شیفت بدهیم که این کار در این کد انجام شده است.

```
(gdb) start
Temporary breakpoint 1 at 0x8390
Starting program: /home/pi/session04/asm/mul03

Temporary breakpoint 1, 0x00008390 in main ()
(gdb) stepi
0x00008394 in main ()
(gdb) stepi
0x00008398 in main ()
(gdb) stepi
0x4005781c in __libc_start_main () from /lib/arm-linux-gnueabi/libc.so.6
(gdb) stepi
0x400722a8 in exit () from /lib/arm-linux-gnueabi/libc.so.6
(gdb) info registers r0 r1
r0          0x28      40
r1          0x5       5
(gdb)
```

تصویر ۱۲: نمایش مقادیر register های کد mul03 با استفاده از GDB

```
pi@raspberrypi ~/session04/asm $ as -o ./mul03.o ./mul03.s
pi@raspberrypi ~/session04/asm $ gcc -o mul03 ./mul03.o
^[[Api@raspberrypi ~/session04/asm $ ./mul03
pi@raspberrypi ~/session04/asm $ echo $?
40
pi@raspberrypi ~/session04/asm $
```

تصویر ۱۳: خروجی کد mul03

در دو تصویر بالا کد mul03 تست شده و مقادیر register های مورد استفاده و خروجی کد نمایش داده شده است.

۳- برنامه ای به زبان اسمبلی بنویسید که دو خانه از حافظه با مقدار اولیه را بخواند و اگر عدد اول بزرگتر از عدد دوم بود ۱ در غیر این صورت ۲ را در تابع main برگرداند


```

GNU nano 2.2.6      File: compare01.s

.data
.balign 4
value1:
    .word 20
.balign 4
value2:
    .word 15

.text
.balign 4
.global main

main:
    LDR r1 , address_value1
    LDR r1 , [r1]
    LDR r2 , address_value2
    LDR r2 , [r2]
    CMP r1 , r2
    BGT value1_max
    mov r0 , #2
    b value2_max
value1_max:
    mov r0 , #1
value2_max:
    bx lr

^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Page ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is  ^U Next Page ^U UnCut Text ^T To Spell

```

تصویر ۱۴: قسمت اول کد compare01

```

GNU nano 2.2.6      File: compare01.s

    LDR r1 , address_value1
    LDR r1 , [r1]
    LDR r2 , address_value2
    LDR r2 , [r2]
    CMP r1 , r2
    BGT value1_max
    mov r0 , #2
    b value2_max
value1_max:
    mov r0 , #1
value2_max:
    bx lr

address_value1 : .word value1
address_value2 : .word value2

```

تصویر ۱۵: قسمت دوم کد compare01

به دلیل عدم امکان نمایش کامل و یک جا کد در QEMU کد در دو عکس جداگانه آورده شده است. دو تصویر ۱۴ و ۱۵ کل کد قسمت اول را نمایش می دهند.

```
Starting program: /home/pi/session04/asm/compare01

Temporary breakpoint 1, 0x00008390 in main ()
(gdb) stepi
0x00008394 in main ()
(gdb) stepi
0x00008398 in main ()
(gdb) stepi
0x0000839c in main ()
(gdb) stepi
0x000083a0 in main ()
(gdb) stepi
0x000083a4 in main ()
(gdb) stepi
0x000083b0 in value1_max ()
(gdb) stepi
0x000083b4 in value2_max ()
(gdb) stepi
0x4005781c in __libc_start_main () from /lib/arm-linux-gnueabi/libc.so.6
(gdb) stepi
0x400722a8 in exit () from /lib/arm-linux-gnueabi/libc.so.6
(gdb) info registers r0 r1 r2 r3 r4 r5 r6
r0             0x1      1
r1             0x14     20
r2             0xf      15
r3             0x8390   33680
r4             0x0      0
r5             0x0      0
r6             0x82e4   33508
(gdb)
```

تصویر ۱۶: نمایش مقادیر register های کد compare01 با استفاده از GDB برای نمونه خروجی اول

```
pi@raspberrypi ~/session04/asm $ as -o ./compare01.o ./compare01.s
pi@raspberrypi ~/session04/asm $ gcc -o compare01 ./compare01.o
pi@raspberrypi ~/session04/asm $ ./compare01
pi@raspberrypi ~/session04/asm $ echo $?
1
pi@raspberrypi ~/session04/asm $ _
```

تصویر ۱۷: نمونه خروجی اول کد compare01

در تصویر ۱۶ و ۱۷ مقادیر register ها و خروجی کد به ازای دو مقدار اولیه به ترتیب ۲۰ و ۱۵ که در کد هم قابل مشاهده است (value1 و value2) نشان داده شده است. به دلیل این که value1 بیشتر است پس مقدار ۱ در خروجی نمایش داده شده است.

```

Temporary breakpoint 1, 0x00008390 in main ()
(gdb) stepi
0x00008394 in main ()
(gdb) stepi
0x00008398 in main ()
(gdb) stepi
0x0000839c in main ()
(gdb) stepi
0x000083a0 in main ()
(gdb) stepi
0x000083a4 in main ()
(gdb) stepi
0x000083a8 in main ()
(gdb) stepi
0x000083ac in main ()
(gdb) stepi
0x000083b4 in value2_max ()
(gdb) stepi
0x4005781c in __libc_start_main () from /lib/arm-linux-gnueabi/libc.so.6
(gdb) stepi
0x400722a8 in exit () from /lib/arm-linux-gnueabi/libc.so.6
(gdb) info registers r0 r1 r2 r3 r4 r5 r6
r0             0x2      2
r1             0xa      10
r2             0xf      15
r3             0x8390   33680
r4             0x0      0
r5             0x0      0
r6             0x82e4   33508
(gdb)

```

تصویر ۱۸: نمایش مقادیر register های کد compare01 با استفاده از GDB برای نمونه خروجی دوم

```

pi@raspberrypi ~/session04/asm $ as -o ./compare01.o ./compare01.s
pi@raspberrypi ~/session04/asm $ gcc -o compare01 ./compare01.o
pi@raspberrypi ~/session04/asm $ ./compare01
pi@raspberrypi ~/session04/asm $ echo $?
2
pi@raspberrypi ~/session04/asm $

```

تصویر ۱۹: نمونه خروجی دوم کد compare01

در تصویر ۱۸ و ۱۹ مقادیر register ها و خروجی کد به ازای دو مقدار اولیه این بار به ترتیب ۱۰ و ۱۵ (value1 و value2) نشان داده شده است. به دلیل این که value2 بیشتر است پس مقدار ۲ در خروجی نمایش داده شده است. مقادیر ۱۰ و ۱۵ در register های r1 و r2 ذخیره شده اند در نمونه خروجی اول هم به همین شکل است.

۴- برنامه ای به زبان C بنویسید که کاری که برنامه های mul01.s و compare01.s انجام می دهند را انجام دهند آن ها را کامپایل و اجرا کنید. توسط GDB فایل های اجرایی را disassemble کنید و با فایل های اسمبلی خود مقایسه کنید.



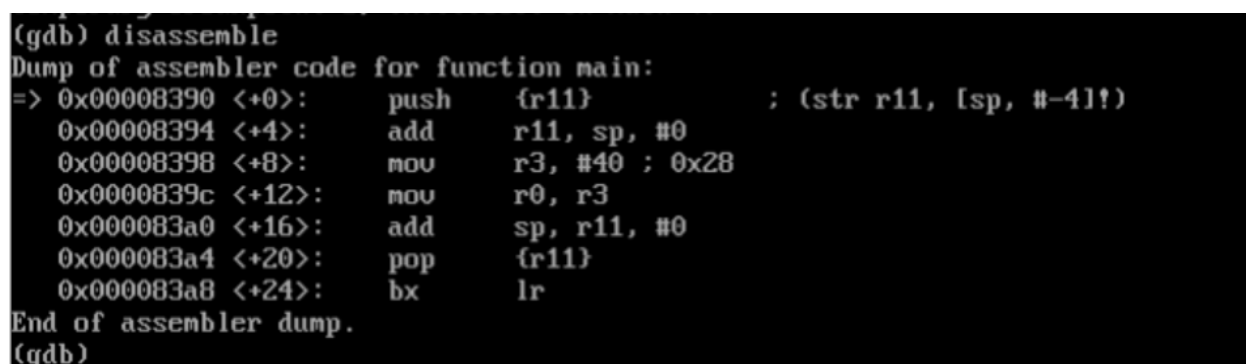
```
GNU nano 2.2.6 File: mul01.c

#include <stdio.h>
int main(){
    return 5 * 8;
}

[ Read 4 lines ]
^G Get Help  ^O WriteOut  ^R Read File  ^Y Prev Page  ^K Cut Text   ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is   ^U Next Page  ^U UnCut Text ^T To Spell
```

تصویر ۲۰: کد mul01 به زبان C

کد بالا مقدار ضرب دو عدد ۵ و ۸ را حساب کرده و به عنوان خروجی تابع main برمی گرداند.



```
(gdb) disassemble
Dump of assembler code for function main:
=> 0x00008390 <+0>:    push    {r11}                ; (str r11, [sp, #-4]!)
0x00008394 <+4>:    add     r11, sp, #0
0x00008398 <+8>:    mov     r3, #40 ; 0x28
0x0000839c <+12>:   mov     r0, r3
0x000083a0 <+16>:   add     sp, r11, #0
0x000083a4 <+20>:   pop     {r11}
0x000083a8 <+24>:   bx      lr
End of assembler dump.
(gdb)
```

تصویر ۲۱: کد mul01 بعد از disassemble شدن توسط GDB

```
GNU nano 2.2.6      File: compare01.c

#include <stdio.h>

int main(){
    int value1 = 10 , value2 = 15;
    if(value1 > value2){
        return 1;
    }else{
        return 2;
    }
}

[ Read 10 lines ]
^G Get Help  ^O WriteOut  ^R Read File  ^Y Prev Page  ^K Cut Text   ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is   ^U Next Page  ^U UnCut Text ^I To Spell
```

تصویر ۲۲: کد compare01 به زبان C

کد بالا بعد از مقایسه value1 و value2 که در حافظه قرار دارند مقدار ۱ یا ۲ را در تابع main برمی گرداند.

```
(gdb) disassemble
Dump of assembler code for function main:
=> 0x00008390 <+0>:      push    {r11}                ; (str r11, [sp, #-4]!)
    0x00008394 <+4>:      add     r11, sp, #0
    0x00008398 <+8>:      sub     sp, sp, #12
    0x0000839c <+12>:     mov     r3, #10
    0x000083a0 <+16>:     str     r3, [r11, #-8]
    0x000083a4 <+20>:     mov     r3, #15
    0x000083a8 <+24>:     str     r3, [r11, #-12]
    0x000083ac <+28>:     ldr     r2, [r11, #-8]
    0x000083b0 <+32>:     ldr     r3, [r11, #-12]
    0x000083b4 <+36>:     cmp     r2, r3
    0x000083b8 <+40>:     ble     0x83c4 <main+52>
    0x000083bc <+44>:     mov     r3, #1
    0x000083c0 <+48>:     b       0x83c8 <main+56>
    0x000083c4 <+52>:     mov     r3, #2
    0x000083c8 <+56>:     mov     r0, r3
    0x000083cc <+60>:     add     sp, r11, #0
    0x000083d0 <+64>:     pop     {r11}
    0x000083d4 <+68>:     bx      lr
End of assembler dump.
(gdb)
```

تصویر ۲۳: کد compare01 بعد از disassemble شدن توسط GDB

در دو کد C این قسمت از `printf` برای نمایش عدد خروجی در کنسول استفاده نشده و با توجه به صورت آزمایش این مقدار در تابع `main` برگردانده شده است. حاصل `disassemble` شده با کد های اسمبلی که در قسمت های قبل برای `mul01` و `compare01` نوشته شده بود تفاوت هایی دارد.

۵- برنامه `stack` را با پیاده سازی به روش `linked list` به زبان C بنویسید.

```
#include <stdio.h>

#include <stdlib.h>

struct node
{
    int data;
    struct node *link;
}*top = NULL;

#define MAX 5

void push();
void pop();
void empty();
void stack_full();
void stack_count();
int st_count();
void destroy();
void print_top();

void main()
{
    int choice;

    while (1)
    {
        printf("\n1. push an element \n");
        printf("\n2. pop an element \n");
```

```
printf("3. check if stack is empty \n");

printf("4. check if stack is full \n");

printf("5. count/display elements present in stack \n");

printf("6. empty and destroy stack \n");

printf("7. Print top of the stack \n");

printf("8. exit \n");

printf("Enter your choice \n");

scanf("%d",&choice);

switch (choice)
{
case 1:
    push();
    break;
case 2:
    pop();
    break;
case 3:
    empty();
    break;
case 4:
    stack_full();
    break;
case 5:
    stack_count();
    break;
case 6:
    destroy();
    break;
case 7:
    print_top();
    break;
case 8:
    exit(0);
```

```
        default:
            printf("wrong choice\n");
        }
    }
}
```

```
void push()
{
    int value , count;

    struct node *temp;
    temp = (struct node*)malloc(sizeof(struct node));

    count = st_count();
    if (count <= MAX - 1)
    {
        printf("\nEnter Your Value :\n");
        scanf("%d",&value);
        temp->data = value;
        temp->link = top;
        top = temp;
    }
    else
        printf("Stack Is Full!\n");
}
```

```
void pop()
{
    struct node *temp;
    if (top == NULL)
        printf("Stack Is Empty\n");
    else
    {
        temp = top;
```



```
    printf("Value Popped Is %d \n",temp->data);
    top = top->link;
    free(temp);
}
}
```

```
void empty()
{
    if (top == NULL)
        printf("Stack Is Empty!\n");
    else
        printf("Stack Is NOT Empty \n");
}
```

```
void stack_full()
{
    int count;

    count = st_count();
    if (count == MAX)
    {
        printf("Stack Is Full\n");
    }
    else
        printf("Stack Is NOT Full \n");
}
```

```
void stack_count()
{
    int count = 0;
    struct node *temp;

    temp = top;
```

```

while (temp != NULL)
{
    printf(" %d\n",temp->data);
    temp = temp->link;
    count++;
}
printf("Size of Stack = %d \n",count);
}

```

```

int st_count()
{
    int count = 0;
    struct node *temp;
    temp = top;
    while (temp != NULL)
    {
        temp = temp->link;
        count++;
    }
    return count;
}

```

```

void destroy()
{
    struct node *temp;
    temp = top;
    while (temp != NULL)
    {
        pop();
        temp = temp->link;
    }
    printf("Stack Destroyed\n");
}

```

```

void print_top()
{
    if (top == NULL)

        printf("\nThere is no Top in an EMPTY stack\n");

    else

        printf("\nTop of the Stack = %d \n",top->data);
}

```

به دلیل طولانی بودن کد و تکه تکه نشدن عکس های کد متن کد به شکل کامل در بالا قرار داده شد.

```

gcc -std=c99 stack.c && a.exe

1. push an element
2. pop an element
3. check if stack is empty
4. check if stack is full
5. count/display elements present in stack
6. empty and destroy stack
7. Print top of the stack
8. exit
Enter your choice
1

Enter Your Value :
10

1. push an element
2. pop an element
3. check if stack is empty
4. check if stack is full
5. count/display elements present in stack
6. empty and destroy stack
7. Print top of the stack
8. exit
Enter your choice
1

Enter Your Value :
80

1. push an element
2. pop an element
3. check if stack is empty
4. check if stack is full
5. count/display elements present in stack
6. empty and destroy stack
7. Print top of the stack
8. exit
Enter your choice
5
80
10
Size of Stack = 2

1. push an element
2. pop an element
3. check if stack is empty
4. check if stack is full
5. count/display elements present in stack
6. empty and destroy stack
7. Print top of the stack
8. exit
Enter your choice

```

تصویر ۲۴: خروجی و تست کوتاهی از کد stack که با linked list پیاده سازی شده است