دستور کار سوم آزمایشگاه ریزپردازنده و زبان اسمبلی

امیرحسین ادواری 98243004 – زهرا حیدری 98243020

سوالات تحلیلی

1-

4 نوع وقفه داریم :

1- Interrupt requests (IRQs) : این نوع وقفه ها Asynchronous میباشند در واقع به کدی که در حال اجراست ارتباطی ندارد . یک سیگنال به prossesor فرستاده می شود و فرایند در حال اجرا به صورت فوری اجرا میشود و interrupt handler اجرا می شود .

2- Non-Maskable interrupts(NMI) : این نوع وقفه ها شبیه به interrupt requests هستند با این تفاوت که غیر قابل چشم پوشی اند . (non-maskable).در شرایطی که مشکل غیرقابل حل در سیستم پیش بیاید رخ می دهد .

3- Execption, faults, software interrupts (generated by core) : این نوع وقفه ها توسط هسته پردازنده ایجاد می شود . مثل اجرا شدن instruction های خاص که موجب سریز شدن در اثر اجرا می‌شوند.

4- Systick Timer : این نوع وقفه ها با زمان اجرای مشخص هستند . source clock این قطعه (systick timer) همان source clock cortex – m cpu می باشد .

وقفه های مختلف دارای اولویت های متفاوت هستند . اگر هم زمان چند وقفه رخ دهند وقفه ای که دارای عدد اولویت کوچکتر است اولویت بالا تری دارد در پردازنده .

-2

اساسی ترین تفاوت آنها در سرعتشان و بهره وری می باشد . سرعت در وقفه بیشتر است چون فرایند سخت افزاری است . اما در سرکشی سرعت کمتر است چون باید چک کنیم ببنیم فرایند مورد نظر اتفاق افتاده یا خیر . در سرکشی هر چقدر نیاز ما به پاسخ سریع تر باشد نیاز به چک کردن هم بیشتر می شود بنابراین زمان بسیار زیادی صرف می شود . اما در وقفه به دلیل سرعت بالاتر بهره وری زمانی آن به طور قابل توجهی بهتر است .

تفاوت دیگر آنها هزینه توسعه آنها می باشد در وقفه ها چون به صورت مستقل و غیر وابسته نسبت به بقیه ی اجزای کد اجرا میشود امکان توسعه ی بخش های مختلف را به طور همزمان دارد و هزینه توسعه آن کمتر است . ولی در سرکشی اجرای وقفه چون به فرایندهای دیگر وابسته است امکان توسعه چند بخش به صورت هم زمان امکان پذیر نیست در مقیاس بزرگ . عیب روش سرکشی بدلیل نرم افزاری بودن این روش اشغال شدید منابع CPU و سایر منابع سیستم است و روش سرکشی آشکار سازی رویداد گسسته زمانی است، و امکان آشکار سازی Real Time را از بین می برد. حال آنکه در روش وقفه، عمل سرکشی با سخت افزار جدا انجام شده و منابعی از سیستم را اشغال نمی کند و مهمتر اینکه امکان آشکار سازی رویداد پیوسته زمانی و Real Time را فراهم می کند.

-3

بردار وقفه حافظه ای می باشد که در آن آدرس ابتدای ISR نوشته و ذخیره می شود . همچنین جا به جایی بردار وقفه نیز به کمک یک register قابل برنامه ریزی به نام vector table offset register انجام می شود .

بردار وقفه در 3 مورد استفاده می شود : 1 – ذخیره کردن برنامه ها در ram 2– تغییر داینامیک بردار و قفه در memory 3– برنامه هایی که boot loader دارند .

-4

وقتی وقفه رخ می دهد همراه با LR ، EXC_RETURN کد نیز ذخیره می شود که CPU آن را تولید می کند . در این register اطلاعاتی مثل اینکه از کدام sp باید register ها بازخوانی شوند و هم چنین این که به کدام مد باید برگردیم و با بیت های 4 تا 7 نیز این که از واحد floating point استفاده کردیم یا خیر ذخیره می شوند . از زمان وقوع وقفه تا زمان شروع اجرای اولین روتین فقط 12 تا سایکل داریم اگر تأخیر سیستم حافظه را صفر در نظر بگیریم و bus بتواند stacking و Fetch vector را همزمان انجام دهد(اسلاید 13 لکچر 8).

https://community.arm.com/arm-community-blogs/b/architectures-and-processors-blog/posts/beginner-guide-on-interrupt-latency-and-interrupt-latency-of-the-arm-cortex-m-processors

https://www.sciencedirect.com/topics/engineering/exception-return-mechanism

+اسلایدهای استاد

بخش عملی

از GPIOA برای تمامی خروجی‌های مورد نیاز، و از GPIOB برای ورودی‌ها وخط اینتراپت استفاده می‌کنیم.

ورودی‌خروجی ها:

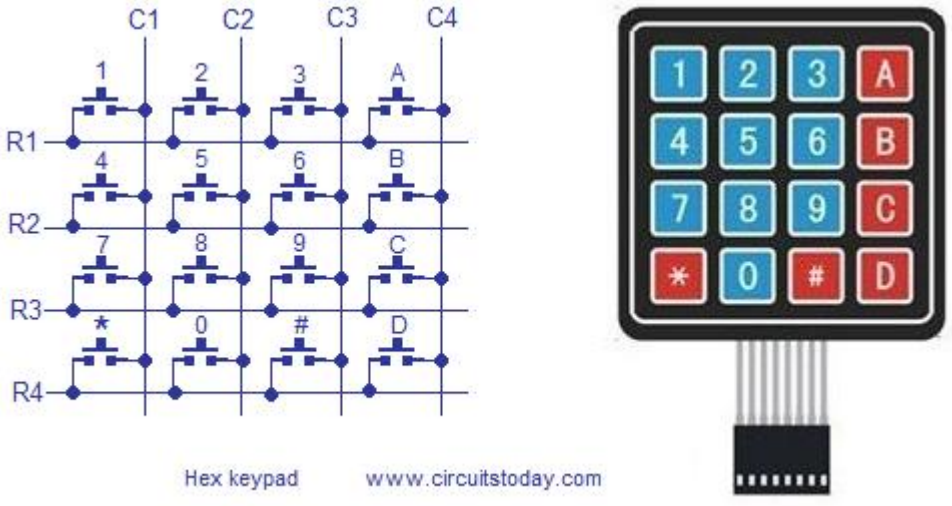
GPIOA:

Pin	PA14	PA13	PA12	PA11	PA10	PA9	PA8	PA7	PA6	PA5	PA4	PA3	PA2	PA1	PA0
Type	OUT	OUT	OUT	OUT	OUT	OUT	OUT	OUT	OUT	OUT	OUT	OUT	OUT	OUT	OUT
Signal	C4	C3	C2	C1	E	RW	RS	D7	D6	D5	D4	D3	D2	D1	D0

GPIOB:

Pin	PB0	PB1	PB2	PB3	PB4	PB5	PB6
Type	(EXTI0)IN	IN	IN	IN	IN	IN	IN
Signal	I	R1	R2	R3	R4	B1	B2

میدانیم ساختار Keypad به شکل زیر است:



برای اینکه متوجه‌شویم کدام کلید فشرده شده است، متناوبا روی C1 تا C4 به صورت متوالی 1 قرار می‌دهیم، و مقادیر R1 تا R4 را چک می‌کنیم (آن‌ها را به خط اینتراپت وصل میکنیم تا درصورت فشرده شدن

دکمه، وقفه صادر شود) با این حساب با هنگام صدور وقفه متوجه می‌شویم یکی از دکمه‌ها فشرده شده است و با علم به اینکه در لحظه صدور روی کدام ستون 1 قرارداده بودیم میتوانیم دقیقا بفهمیم کدام دکمه فشرده شده است.

تمامی ورودی ها، اعم از 4 خط برای R1 تا R4 را بعلاوه دو خط برای B1 و B2، OR میکنیم و وارد خط مربوط به تشخیص اینتراپمان میکنیم. تا در صورت فشردن هر کدام از آن‌ها وقفه صادر شود، سپس با روالی که بالاتر ذکر شد، بررسی میکنیم کدام کلید دقیقا فشرده شده‌است. اگر هیچکدام از کلیدهای keypad فشرده نشده اما وقفه صادر شده است پس B1 یا B2 فشرده شده‌است.

در این بخش قسمت‌های مختلف کد را جداگانه شرح می‌دهیم :

- **Defines:**

```
#include <stm32f4xx.h>
#include <stdbool.h>

// GPIOA -> For outputs :
#define RS (8)
#define RW (9)
#define E  (10)
#define C1 (11)
#define C2 (12)
#define C3 (13)
#define C4 (14)

// GPIOB -> For inputs
#define R1 (1)
#define R2 (2)
#define R3 (3)
#define R4 (4)
#define B1 (5)
#define B2 (6)

#define MASK(x)    (1UL << (x))
```

ورودی‌خروجی هارا دیفاین می‌کنیم(D0 تا D7 را قرار ندادیم زیرا آن‌ها را به ترتیب در 8 بیت پایین PA قرار دادیم لذا نوشتن آن‌ها سهل است.)

- **commitChar**:

```
void commitChar(char data){
    GPIOA->ODR &= 0x00;
    GPIOA->ODR |= MASK(E);

    GPIOA->ODR &= ~MASK(RW);
    GPIOA->ODR |= MASK(RS) | data;
    GPIOA->ODR &= ~MASK(E);
    delayms(1);
}
```

این تابع طبق جداول داده شده و ورودی خروجی‌های تعریف شده، کاراکتر ورودی را در مکان فعلی اشاره‌گر lcd مینویسد.

- **commitCommand:**

```
void commitCommand(char command){
    GPIOA->ODR &= 0x00;
    GPIOA->ODR |= MASK(E);

    GPIOA->ODR &= ~MASK(RW);
    GPIOA->ODR &= ~MASK(RS);
    GPIOA->ODR |= command;
    GPIOA->ODR &= ~MASK(E);
    delayms(1);
}
```

این تابع کد دستوری داده شده را به lcd اعمال میکند، RS و RW را مشخصا 1 کرده، کامند را در D0 تاD7 نوشته؛ و یک پالس روی E ایجاد میکند تا تغییر اعمال شود. کمی نیز تاخیر ایجاد می‌کند.

- **variable Declarations:**

```
volatile int currentCol = 0;
volatile bool firstCharPresent = false;
volatile bool isSecondOperandPresent = false;
volatile bool isFirstOperandPresent = false;
volatile int32_t firstOperand = 0;
volatile int32_t secondOperand = 0;
volatile char op = ' ';
volatile bool unaryPresent = false;
```

طبق توضیحات اولیه، متغیر اول نشان میدهد در حال حاضر روی کدام ستون keypad ، یک گذاشته ایم.

متغیر دوم مشخص میکند که آیا اولین کاراکتر روی lcd رفته است یا خیر.

متغیر سوم و چهارم مشخص می‌کنند آیا عملوندهای اول دوم وارد شده‌اند یا خیر.

متغیرهای پنجم و ششم مقادیر عملوندها و متغیر ششم نوع عملیات را تعیین میکند.

متغیر آخر نشان میدهد که آیا آخرین نوشتن در lcd، مربوط به unary ها یعنی inc و dec بوده یا خیر.

- **delayms:**

```
void delayms(uint16_t ms){
    uint32_t counter = 0;
    while(ms > 0) {
        counter = 0;
        while( counter < 1000){
            counter++;
        }
        ms--;
    }
}
```

این تابع صرفا با مشغول کردن پردازنده تاخیر ایجاد می‌کند.

- **init_configs:**

```
void init_configs(){
    // Enable GPIOA And GPIOB:
    RCC -> AHB1ENR |= RCC_AHB1ENR_GPIOAEN;
    RCC -> AHB1ENR |= RCC_AHB1ENR_GPIOBEN;

    // Set Moder For GPIOA (14 outputs)
    GPIOA->MODER = 0x55555555;

    // Set Moder For GPIOB
    GPIOB->MODER &= 0xFFFFC000;

    // Pull-down for inputs:
    GPIOB->PUPDR = 0x00002AAA;


    RCC->APB2ENR |= RCC_APB2ENR_SYSCFGEN;

    // Config Interrupt line
    SYSCFG->EXTICR[0] |= SYSCFG_EXTICR1_EXTI0_PB;


    EXTI->IMR |= MASK(0);
    EXTI->RTSR |= MASK(0);
    __enable_irq();

    NVIC_ClearPendingIRQ(EXTI0_IRQn);
    NVIC_EnableIRQ(EXTI0_IRQn);

}
```

این تابع، در ابتدا، GPIOA و GPIOB را فعال میکند، سپس مدر را با توجه به ورودی‌خروجی های تعیین شده در ابتدای همین گزارش ست میکند (01 برای خروجی و 00 برای ورودی) سپس اینتراپت را روی PB0 ست کرده، روی لبه بالارونده گذاشته و بیت متناظر آن در اینتراپت مسک رجیستر را ست میکند. تابع مربوط به فعال کردن گلوبال همه اینتراپت‌هارا کال کرده و در نهایت اینتراپت‌های در حالت pending را در NVIC پاک کرده و اینتراپت خروجی مربوطه را در آن فعال مینماید.

- **init_lcd:**

```
void init_lcd(){
    // Turn On :
    commitCommand(0x0E);

    // 8-bit 2 line mode :
    commitCommand(0x38);
}
```

این تابع صرفا LCD را روشن کرده و روی مد دوخطی و 8 بیتی قرار میدهد.

- **printDigit:**

```
void printDigit(int N){
        if( N == 0 ){
            printInLine2(intToChar(N));
            return;
        }
    char arr[30];
    int i = 0;
    int j, r;

    while (N != 0) {
        r = N % 10;
        arr[i] = r;
        i++;
        N = N / 10;
    }
    for (j = i - 1; j > -1; j--) {
        printInLine2(intToChar(arr[j]));
    }
}
```

این تابع یک عدد چندرقمی گرفته، و آنرا رقم به رقم از سمت چپ روی LCD مینویسد. روال کار اینگونه است که ابتدا با باقیمانده گرفتن و تقسیم بر 10 کردن، از سمت راست همه ارقام را در یک آرایه میریزد، سپس آرایه را از اخر میپیماید و روی LCD مینویسد.

- **intToChar, CharToInt:**

```
char intToChar(int32_t digit){
        switch(digit){
        case 0: return '0';
        case 1: return '1';
        case 2: return '2';
        case 3: return '3';
        case 4: return '4';
        case 5: return '5';
        case 6: return '6';
        case 7: return '7';
        case 8: return '8';
        case 9: return '9';
        default: return '0';
    }
    return '0';
}
int32_t charToInt(char ch){
    switch(ch){
        case '0': return 0;
        case '1': return 1;
        case '2': return 2;
        case '3': return 3;
        case '4': return 4;
        case '5': return 5;
        case '6': return 6;
        case '7': return 7;
        case '8': return 8;
        case '9': return 9;
        default: return 0;
    }
    return 0;
}
```

این دو تابع صرفا لیترال‌های عددی یک تا نه را به حالت کاراکتری آن‌ها و بالعکس تبدیل میکنند.

- **is_binary_operator, is_unary_operator, printInLine2:**

```
void printInLine2(char ch){
    commitChar(ch);
}
bool is_binary_operator(char ch){
    return ch == '+' || ch == '-' || ch == '*' || ch =='/';
}
bool is_unary_operator(char ch){
    return ch == 'i' || ch == 'd';
}
```

تابع پرینت صرفا تابع commitChar را کال میکند که توضیح داده شد، دو تابع دیگر صرفا بررسی میکنند که کاراکتر ورودی چه نوع اپراتوری است.

- **PrintLine:**

```
void printInLine(char* string){
    while(*string != '\0'){
        char ch = (char) (*string);
        commitChar(ch);
        string++;
    }
    commitCommand(0xC0);
}
```

این تابع یک رشته را کاراکتر به کاراکتر روی خط فعلی lcd نوشته و درنهایت اشاره‌گر را به ابتدای خط دوم lcd می‌برد.

- **clear_line_2:**

```
void clear_line_2(){
    commitCommand(0xC0);
    printInLine("                ");
}
```

این تابع صرفا خط دوم lcd را پاک می‌کند.(0XC0 پوینتر را به ابتدای خط دوم میبرد، با چاپ 16 فاصله، کل خط دوم پاک می‌شود)

- **printUnaryOperator:**

```
void printUnaryOperator(int32_t operand, bool increment){
    int32_t operandCopy = operand;

    if(unaryPresent && increment){
            operand--;
    }
    if(unaryPresent && !increment) {
        operand++;
        commitCommand(0x10);
        commitChar(' ');
    }
    if(operand == 0)
        commitCommand(0x10);

    while(operand != 0){
        operand /= 10;
        // move cursor left
        commitCommand(0x10);
    }
    if(unaryPresent){
        commitCommand(0x10);
        commitCommand(0x10);
        commitCommand(0x10);
        commitCommand(0x10);
        commitCommand(0x10);
    }
    if( operandCopy < 0)
        commitCommand(0x10);

    if(increment){
        printInLine2('i');
        printInLine2('n');
        printInLine2('c');
    }
    else {
        printInLine2('d');
        printInLine2('e');
        printInLine2('c');
    }
    printInLine2('(');
    printDigit(operandCopy);
    printInLine2(')');
}
```

این تابع ورودی درون LCD را که روی آن عملگر تک‌عملوندی اعمال شده است را پاک کرده و آنرا درون inc(x) یا dec(x) قرار داده و روی lcd چاپ می‌کند. همچنین اینکه روی یک عملگر تک عملوندی یک عملگر تک عملوندی دیگر اعمال شود را نیز به کمک بولین unaryPresent هندل میکند.

- **unary_click:**

```
char unary_click(){
    if( GPIOB->IDR & MASK(B1)){
        while(GPIOB->IDR & MASK(B1));
        return 'i';
    }
    else if(GPIOB->IDR & MASK(B2)){
        while(GPIOB->IDR & MASK(B2));
        return 'd';
    }
}
```

این تابع بررسی می‌کند آیا دو کلید مربوط به عملگر های تک‌عملوندی فشرده شده اند یا نه، و سپس کاراکتر مربوط به آن‌ها را که در سایر توابع استفاده میشود را برمیگرداند.

کاراکتر i بعنوان increment و کاراکتر d بعنوان decrement اطلاق می‌شود.

- **clickedButton:**

```
char clicked_button(){

    if( GPIOB->IDR & MASK(R1)) {
        while( GPIOB->IDR & MASK(R1));
        switch(currentCol){
            case 0: return '7';
            case 1: return '8';
            case 2: return '9';
            case 3: return '/';
        }
    }
    if( GPIOB->IDR & MASK(R2)) {
        while(GPIOB->IDR & MASK(R2));
        switch(currentCol){
            case 0: return '4';
            case 1: return '5';
            case 2: return '6';
            case 3: return '*';
        }
    }
    if( GPIOB->IDR & MASK(R3)) {
        while(GPIOB->IDR & MASK(R3));
        switch(currentCol){
            case 0: return '1';
            case 1: return '2';
            case 2: return '3';
            case 3: return '-';
        }
    }
    if( GPIOB->IDR & MASK(R4)) {
        while(GPIOB->IDR & MASK(R4));
        switch(currentCol){
            case 0: return 'C';
            case 1: return '0';
            case 2: return '=';
            case 3: return '+';
        }
    }
    return unary_click();
}
```

این تابع طبق روال شرح داده شده در ابتدای گزارش تعیین میکند کدام دکمه منجر به صدور وقفه شده-است (این تابع درون اینتراپت هندلر کال می‌شود)

- **calculate:**

```
void calculate(){
    switch(op){
        case '+':
            firstOperand += secondOperand;
            break;
        case '-':
            firstOperand -= secondOperand;
            break;
        case '/':
            firstOperand /= secondOperand;
            break;
        case '*':
            firstOperand *= secondOperand;
            break;
    }
    isFirstOperandPresent = true;
    isSecondOperandPresent = false;
    secondOperand = 0;
    firstCharPresent = true;
    op = ' ';
    printDigit(firstOperand);
}
```

این تابع زمانی کال می‌شود که تمامی ملزومات برای انجام یک عملیات دو عماوندی مهیا باشد که عبارتند از firstOperand، secondOperand و نیز نوع عملیات یا همان op . سپس مطابق با این 3 مورد عملیات مورد نظر انجام می‌شود و حاصل آن در firstOperand ریخته می‌شود(همانند عملکرد ماشین حساب ویندوز) سایر متغیرهای کنترلی را متناسب با اینکه اکنون firstOperand را داریم ست می‌کند.

- **main:**

```
int main(void){

    init_configs();
    init_lcd();
    printInLine("Advari-Heidari");
    printInLine("Welcome");
    delayms(10000);
    clear_line_2();

    // Main Loop of program:
    char cols[] = { C1, C2, C3, C4};
    while(1){
        GPIOA->ODR &= ~MASK(cols[currentCol]);
        currentCol =(currentCol+1) % 4;
        GPIOA->ODR |= MASK(cols[currentCol]);
        delayms(1);
    }
}
```

در تابع main عملا کانفیگ‌های شرح داده شده را انجام می‌دهیم، سپس در خط اول lcd نام‌هارا نوشته و در خط دوم یک عبارت چاپ می‌کنیم و بعد از مدتی پاک میکنیم تا ماشین حساب آماده استفاده شود.

سپس در یک لوپ متناوبا ستون‌های keypad را یک میکنیم که جزییات آن شرح داده شد.

- **InterruptHandler:**

```
void EXTI0_IRQHandler(void){
    EXTI->PR |= MASK(0);
    NVIC_ClearPendingIRQ(EXTI0_IRQn);
    char clickedButton = clicked_button();
```

در اینتراپت هندلرمان، ابتدا بیت پندیگ‌رجیستر مربوط به وقفه کنونی را ست کرده، و آنرا در NVIC نیز از حالت Pending در می‌آوریم. سپس کاراکتر دکمه فشرده شده را بدست می‌آوریم (توابع شرح داده شدند)

```
if (!is_unary_operator(clickedButton) && !is_binary_operator(clickedButton) ){
    unaryPresent = false;
}
```

در این خط بررسی میکنیم اگر کلید فشرده شده، اپراتور نیست پس unaryPresent فالس می‌شود (بعد عملگر تک عملوندی فقط آمدن عدد غیر مجاز است)

```
if( clickedButton == 'C') {
    firstCharPresent = false;
    isSecondOperandPresent = false;
    isFirstOperandPresent = false;
    firstOperand = 0;
    secondOperand = 0;
    unaryPresent = false;
    op = ' ';
    clear_line_2();
}
else if( clickedButton == '='){
    clear_line_2();
    calculate();
}
```

اگر دکمه فشرده شده C باشد، بایستی همه متغیرهای کنترلی ریست شوند و صفحه پاک شود.

اگر دکمه فشرده شده مساوی باشد، یعنی عملوندها و عملیات مشخص شده است و بایستی عملیات انجام شود لذا روتین calculate را کال می‌کنیم.

```
else if( is_binary_operator(clickedButton) ){
    if( !firstCharPresent ){
        if(clickedButton == '-'){
            firstOperand = 0;
            isFirstOperandPresent = true;
            firstCharPresent = true;
            op = '-';
            printInLine2('0');
            printInLine2('-');
        }
    }
    else {
        if(isSecondOperandPresent){
            clear_line_2();
            calculate();
        }
        else{
            isFirstOperandPresent = true;
            if( op != ' ')
                commitCommand(0x10);
        }
        op = clickedButton;
        printInLine2(op);
    }

}
```

اگر دکمه ورودی، یک اپراتور دوعملوندی است (–و +و *و /) بایستی بررسی شود که آیا اولین کاراکتر وارد شده میباشد یا خیر، اگر اولین کاراکتر باشد لزوما میتواند – باشد که نشانگر علامت منفی‌است. با این حساب اگر عملگر منفی بود، بایستی عملوند اول صفر شده و طی یک عملیات تفریق منتظر عملوند دوم باشیم (x- را عملا 0-x در نظر گرفتیم که دقیقا مشابه ماشین حساب ویندوز است)

اگر ورودی اپراتور دو عملوندی بود و اولین کاراکتر نبود، بررسی میکنیم که آیا عملوند دوم آمده یا نه، اگر آمده باشد یعنی این عملگر برای عملیات بعدی‌است و عملیات فعلی هنوز محاسبه نشده است، لذا خط را پاک کرده و عملیات قبلی را محاسبه کرده و مینویسیم. اما اگر عملوند دوم نیامده باشد، یعنی این عملگر دقیقا بعد عملوند اول آمده (عملوند اول حاضر است) پس این عملگر را ست می‌کنیم و اگر قبل آن نیز عملگری آمده

آنرا overwrite می‌کنیم (دقیقا مشابه ویندوز) در نهایت عملگر جدید را ست کرده و آنرا روی خط دوم می-نویسیم تا عملوند بعدی بیاید.

```
else if (is_unary_operator(clickedButton)){
    int32_t operand;
    bool increment;
    if(isSecondOperandPresent){
        if(clickedButton == 'i'){
            operand = secondOperand++;
            increment = true;
        }
        else{
            operand = secondOperand--;
            increment = false;
        }
    }
    else if (op != ' ')
        return;
    else if (!firstCharPresent)
        return;
    else{
        if(clickedButton == 'i'){
            operand = firstOperand++;
            increment = true;
        }
        else{
            operand = firstOperand--;
            increment = false;
        }
    }
    printUnaryOperator(operand, increment);
    unaryPresent = true;
}
```

اگر دکمه فشرده شده یک عملگر تک‌عملوندی می‌باشد، بررسی می‌کنیم که آیا عملوند دومی برای عملیات انجام نشده وجود دارد یا خیر؛ اگر وجود دارد با توجه به اینکه عملگرمان increment یا decrement است، آنرا کم یا زیاد میکنیم و در نهایت مقدار آن‌را درون inc() یا dec() قرار می‌دهیم.

اگر مقدار عملوند دوم حاضر نیست بررسی میکنیم، اگر op تعیین شده است عملگر تک عملوندی را تاثیر نمیدهیم (بعد از یک اپراتور نمیتواند اپراتور تک عملوندی بیاید) در غیر اینصورت همان روال شرح داده شده را برای عملوند اول در صورت وجود انجام می‌دهیم.

```
else if( !isFirstOperandPresent ){
    firstOperand *= 10;
    firstOperand += charToInt(clickedButton);
    if( ! ( clickedButton == '0' && firstOperand == 0 && firstCharPresent ) )
        printInLine2(clickedButton);
    firstCharPresent = true;

}
else{
    isSecondOperandPresent = true;
    secondOperand *= 10;
    secondOperand += charToInt(clickedButton);
    printInLine2(clickedButton);
}
```

در نهایت اگر عملوند اول نیامده است آنرا میخوانیم و با شیفت مناسب عدد فعلی در متغیر مربوطه قرار می‌دهیم.

اگر حالتهای بالا رخ ندهد یعنی کاربر در حال وارد کردن عملوند دوم است.

پروتئوس

طبق توضیحات داده شده مدار را میبندیم :