فاطمه سادات سیفی – 98243035

پارسا محمدپور – 98243050

تحلیلی:

۱-

وقفه های سخت افزاری:

اگر سیگنال پردازنده از دستگاه یا سخت افزار خارجی باشد، وقفه سخت افزاری نامیده می شود. (مثلا از یک دکمه ی خارجی تولید شده باشد) وقفه های سخت افزاری را می توان به دو نوع طبقه بندی کرد:

- **Maskable Interrupts**: یکی از انواع وقفه های سخت افزاری است که در صورتی که وقفه ای با الویت بالاتر رخ دهد قابل تعویق افتادن است.
- **Non-maskable Interrupts (NMI):** یکی از وقفه های سخت افزاری است با این تفاوت که در هر صورت باید به طور فوری رخ دهد و قابل تعویق انداختن نیست.

وقفه های نرم افزاری:

وقفه های نرم افزاری نیز به دو نوع تقسیم می شوند:

- **وقفه های عادی:** وقفه هایی که در اثر دستورالعمل های نرم افزار ایجاد می شوند.
- **استثنا:** وقفه های برنامه ریزی نشده در حین اجرای یک برنامه Exception نامیده می شود.(تقسیم عددی بر ۰ حین اجرای برنامه تولید Exception میکند).

در شرایطی که چند وقفه همزمان رخ دهد پردازنده بر اساس priority وقفه ها تصمیم میگیرد و هرکدام که priority کوچکتری داشته باشد را الویت میدهد.

۲-

در حالت کلی روش سرکشی یا Polliang مزیتی نسبت به روش وقفه تا Interrupt ندارد و وقتی که برای اطلاع از رویدادی امکان استفاده از وقفه را بنا به هردلیلی نداریم (مثلا امکان سخت افزاری وقفه در نظر گرفته نشده و یا منابع وقفه اشغال می باشد) به ناچار از روش سرکشی استفاده می کنیم.
در روش سرکشی cpu و سایر منابع شدیدا اشغال میشوند.در این روش امکان آشکار سازی real time وجود ندارد. برعکس interrupt.
تفاوت اصلی بین وقفه و سرکشی این است که در وقفه، دستگاه به CPU اطلاع می دهد که نیاز به توجه دارد، در حالی که در سرکشی CPU به طور مداوم وضعیت دستگاه ها را بررسی می کند تا بفهمد آیا آنها نیاز به توجه دارند یا خیر.
Interuption زمانی کارآمد است که دستگاه به طور مکرر CPU را قطع کندpolling در فواصل منظم رخ می دهد بیت آماده فرمان نشان می دهد که دستگاه به یک سرویس نیاز دارد وقتی CPU به ندرت درخواست هایی از دستگاه ها پیدا می کند، polling ناکارآمد است Polling هزینه و زمان بیشتری مصرف میکند.

۳-

بردارهای وقفه آدرس هایی هستند که به کنترل کننده وقفه اطلاع می دهند که کجا ISR را بیابند. به همه وقفه ها عددی از ۰ تا ۲۵۵ اختصاص داده می شود که هر یک از این وقفه ها با بردار وقفه خاصی مرتبط هستند. و جابجایی آن توسط programmable vector table register که آدرس ISR درآن ذخیره میشود انجام میشود.

بردار وقفه محل حافظه یک کنترل کننده وقفه است که وقفه ها را اولویت بندی می کند و اگر بیش از یک وقفه در انتظار رسیدگی باشد آنها را در یک صف ذخیره می کند. برای مثال در ذخیره کردن برنامه ها در ram استفاده میشود.

۴- در حین exception handling مقدار LR به مقدار EXC_RETURN به روز رسانی میشود و سپس برای راه اندازی exception return در انتهای exception handler استفاده می شود.

رجیستر EXC_RETURN برای برقراری ارتباط اطلاعات اضافی در مورد اینکه پس از exception handling به چه وضعیتی باید بازگشت و کدام register ها باید از پشته خارج شوند استفاده می شود.

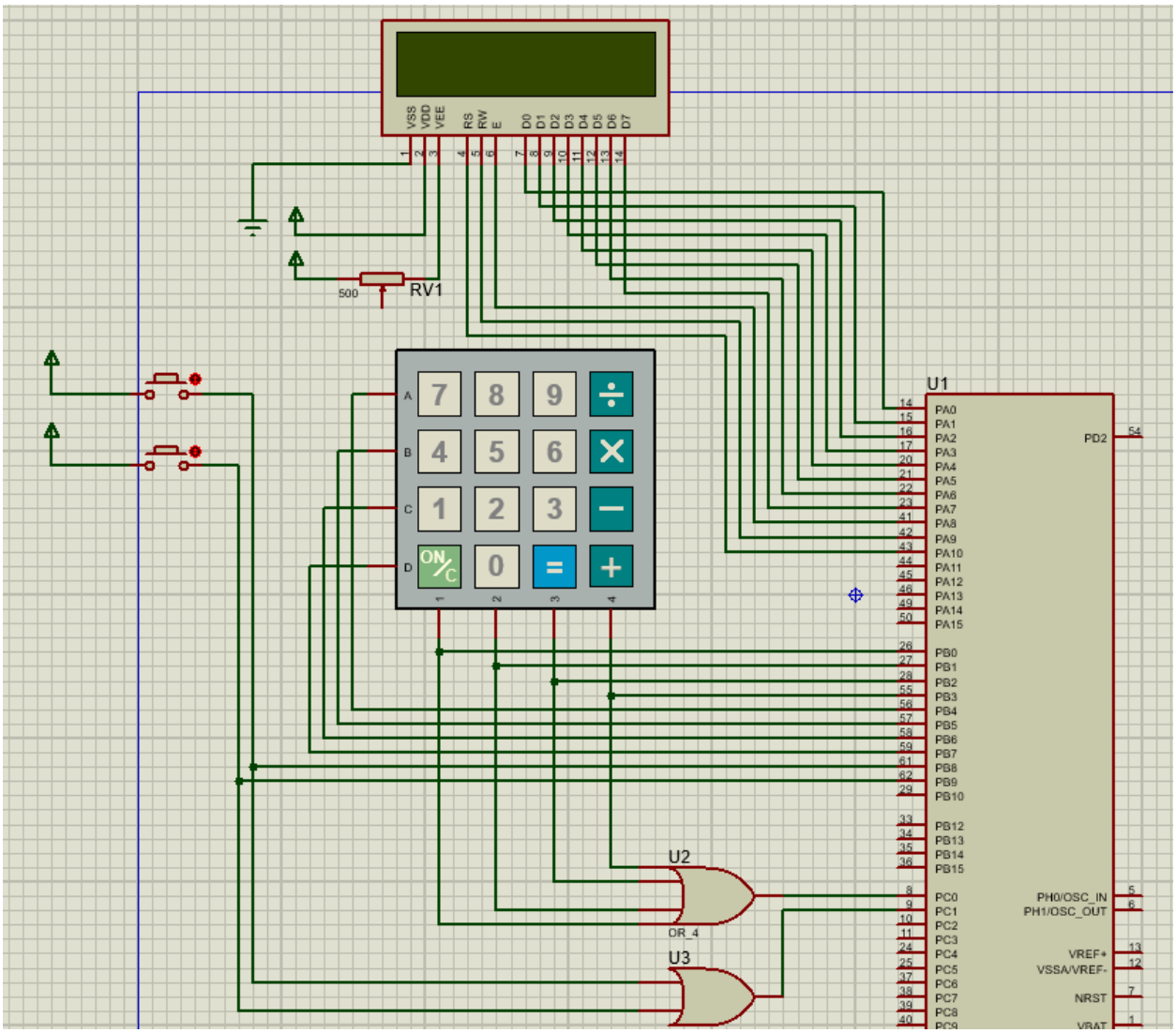
با توجه به اسلاید های درس، تنها 12 سیکل از exception request تا اولین اینستراکشن در handler طول می کشد اگر memory latency سیستم و همچنین bus ساپورت کننده vector fetch و stacking همزمان داشته باشیم.

کدی:

در این قسمت از ما خواسته شده است تا یک ماشین حساب را با استفاده از یک keypad و LCD2*16 و دو عدد button پیاده سازی کنیم، به طوری که هر کدام از این دکمه ها یک واحد به عدد روی LCD اضافه یا از آن کم می کند.

Proteus:

در این قسما شماتیک مدار و نحوه وصل کردن پین ها را بیان میکنیم.



حال با توجه به شکل، هر کدام از پین ها و کارکردشان را بیان میکینم:

پین	ورودی / خروجی میکرو	کاربرد	مقدار MODER هر پین	مقدار PUPDR
PA0	خروجی	D0 در LCD	01	-
PA1	خروجی	D1 در LCD	01	-
PA2	خروجی	D2 در LCD	01	-
PA3	خروجی	D3 در LCD	01	-
PA4	خروجی	D4 در LCD	01	-
PA5	خروجی	D5 در LCD	01	-
PA6	خروجی	D6 در LCD	01	-
PA7	خروجی	D7 در LCD	01	-
PA8	خروجی	E در LCD	01	-
PA9	خروجی	RW در LCD	01	-
PA10	خروجی	RSدر LCD	01	-
PB0	ورودی	خروجی 1 در keypad	00	10
PB1	ورودی	خروجی 2 در keypad	00	10
PB2	ورودی	خروجی 3 در keypad	00	10
PB3	ورودی	خروجی 4 در keypad	00	10
PB4	خروجی	ورودی A در keypad	01	-
PB5	خروجی	ورودی B در keypad	01	-
PB6	خروجی	ورودی C در keypad	01	-
PB7	خروجی	ورودی D در keypad	01	-
PB8	ورودی	خروجی کلید اول	00	10
PB9	ورودی	خروجی کلید دوم	00	10
PC0	ورودی	نشان دهنده تغییر و فشار داده شدن keypad	00	10
PC1	ورودی	نشان دهنده تغییر و فشار داده شدن کلید ها	00	10

در گیت OR قرار داده شده برای این است که اگر هر گونه تغییری در یکی از خروجی های keypad اتفاق افتاد متوجه آن شویم و سپس interrupt routine مخصوص به keypadها را انجام دهیم و گیت OR دیگر هم برای این است که هر گاه یکی از دو کلید فشار داده شد، متوجه تغییر در آن شویم و سپس interrupt routine مخصوص به آن را اجرا کنیم.(در خود interrupt routine ها سازوکار لازم را برای اینکه بفهمیم کدام یکی از ورودی های میکرو مان تغییر کرده است، لحاظ میکنیم. این روش هم آسان تر و کوتاه تر است، هم به دلیل اینکه اگر جدا جدا لحاظ می کردیم، ۶ تا خط interrupt نیاز داشتیم و ست کردن ۶ تا خط اینتراپت سخت و یاحتی به خاطر ساختار درونی آن ناممکن و یا به شدت سخت بود، این حرکت را انجام دادیم.)

Keil:

برای این کار برای اینکه به جای استفاده از اعداد، از اسم های مرتبط استفاده کنیم که کد خوانا تر شود، یک سری define هایی انجام دادیم، همینطور برای کم کردن کر، یک سری MASK هایی را تریف کردیم در همان ابتدا و با توجه به شماره پینی که هر کدام از آن ها به آن متصل است، مقدار دهی کردیم.

سپس اسم تابع های مورد استفاده و ورودی و خروجیشان را مشخص کردیم تا وارنینگ ها بر طرف شوند.

ساز و کار کلی ما به این شکل است که تمامی کاراکتر هایی که می خواهیم بر روی LCD نمایش دهیم، را در یک آرایه قرار میدهیم تا مقدار آنها را بتوانیم هنگامی که می خواهیم عملیاتی انجام دهیم داشته باشیم و سپس برای این آرایه یک عدد size تعریف کردیم که نشان می دهد تا هر لحظه چه تعداد از خانه های این آرایه پر است.(این عدد در هر لحظه به اولین خانه خالی آرایه اشارهع میکند.) سپس یک عدد status هم در نظر گرفتیم که در مواقعی که دکمه ای از keypad فشار داده میشود، اگر آن دکمه مساوی یا clear است بتوانیم بفهمیم. تیکه کد زیر مربوط به این بخش می باشد:

```
#include<stm32f4xx.h>

//Defining a variable to store the port number 7 use them easier
// LCD pins number in port A
#define d0 0 //PA0
#define d1 1 //PA1
#define d2 2 //PA2
#define d3 3 //PA3
#define d4 4 //PA4
#define d5 5 //PA5
#define d6 6 //PA6
#define d7 7 //PA07

#define E 8 //PA8
#define RW 9 //PA9
#define RS 10 //PA10

// Keypad pins number in port B
#define A 4//PB0
#define B 5//PB1
#define C 6//PB2
#define D 7//PB3

#define one 0 //PB4
#define two 1 //PB5
#define three 2 //PB6
#define four 3 //PB7

//Button 1 pins in port B
#define sw1 8 //PB8
#define sw2 9 //PB9

//OR gates for intrrupts in port C
#define or1 0 // PC0
#define or2 1 // PC1

```

```
//MASKS:
#define SET1(x) (1ul << (x))
#define SET0(x) (~(1ul << (x)))

//variables for calculating the sum
static char array[16];
static int size = 0; // return the first empty index
static int status = 0; // 0 = usual      1 = finding result       2 = clear

//Defining Functions
void initialize(void);
void findKeypadButton(void);
void LCD_put_char(char data);
void delay(volatile int n);
void LCD_init(void);
void LCD_command(unsigned char command);
void LCD_resetCommand(void);
void LCD_setCommand(void);
int getNumber(char c);
void inc_dec_number(int value);
char getChar(int digit);
void calculate(void);
void EXTI0_IRQHandler(void);
void EXTI1_IRQHandler(void);
void resetArray(void);
```

حال به سراغ عملکرد برنامه و توضیح کد ها می رویم. برای این کار، باید یک سری کار ها و تنظیمات اولیه را انجام دهیم، (همانطور که این تنظیمات را در یک تابع به اسم initialize قرار دادیم و در main ابتدا قبل از ورود به حلقه آن را فراخوانی کردیم)از جمله تنظیم کردن پین های ورودی و خروجی مشخص شده در جدول بالا (این تنظیمات در همان جدول ذکر شده است) و تنظیمات مربوط به وقفه ها.

تیکه کد زیر، مربوط به روشن کردن clock برای GPIO های A,B و C (در ابتدا clock ها خاموش است تا توان مصرفی کاهش)و همچنین تنظیمات هر پین (بنابر کارکردشان و مقادیر ذکر شده در جدول) می باشد:

```
  RCC -> AHB1ENR |= RCC_AHB1ENR_GPIOAEN; // turning on the clocks for GPIOA
  RCC -> AHB1ENR |= RCC_AHB1ENR_GPIOBEN; // turning on the clocks for GPIOB
  RCC -> AHB1ENR |= RCC_AHB1ENR_GPIOCEN; // turning on the clocks for GPIOC

  //GPIOA is output. So we set the MODE for them (pin 0 to 10) writing mode which is "01". There are 11 pins, so
  //the number will be 01 0101 0101 0101 0101 0101
  GPIOA -> MODER = 0x155555;

  //in GPIOB the first 4 pins are keypads output(micro's input), next 4 pins are keypads input(micro's output) and 2 last pins
  //are buttens output(micro's input). reading MODE : "00"       writing MODE : "01"    => 0101 0101 0101 0000 0000
  GPIOB -> MODER = 0x55500;

  //in GPIOC 2 first pins are input. MODE code for input is "00". So the number will be: 0000
  GPIOC -> MODER = 0x0;

  //in keypad, first 4 pins in GPIOB which are connected to keypads output(micro's input) should be pull-down. and 2 last pins in GPIOB
  //which are connected to buttons, should be pull-down. pull-down : "10"         => 1010 0000 0000 1010 1010
  GPIOB -> PUPDR = 0xA00AA;

  //in GPIOC both input pins are pull-down and pull-down code is "10". So the number will be : 1010
  GPIOC -> MODER = 0xA;
```

سپس تمامی خروجی هایی که از میکرو به keypad وصل می شوند را مقدار دهی کردیم و مقدار آن همه آنها را یک قرار دادیم.

سپس با استفاده از فایل کمکی و جدول مربوط به LCD که در اختیارمان قرار داده شد، ابتدا میایم و مقداردهی های اولیه و تنظیمات LCD را انجام می دهیم، سپس تابع LCD_setCommand را صدا میزنیم (کار این تابع این است که اسم دو نفر را در خط اول LCD می نویسد و سپس نشانگر را به خط دوم می برد). تیکه کد زیر مربوط به این بخش می باشد:

```
  //turning on the system configuration clock
  RCC -> APB2ENR |= RCC_APB2ENR_SYSCFGEN;

  //onnecting intrrupt line 1 and 2 to port B
  SYSCFG -> EXTICR[0] |= SYSCFG_EXTICR1_EXTI0_PC; // keypad
  SYSCFG -> EXTICR[0] |= SYSCFG_EXTICR1_EXTI1_PC; // buttons

  //removing intrrupt mask register for line 1 and 2. And we also set them in rising edge
  EXTI -> IMR |= SET1(0) | SET1(1);
  EXTI -> RTSR |= SET1(0) | SET1(1);

  //turning on the intrrupts line
  __enable_irq();

  //setting priority
  NVIC_SetPriority(EXTI0_IRQn, 0);
  NVIC_SetPriority(EXTI1_IRQn, 0);

  //clearing pending registers
  NVIC_ClearPendingIRQ(EXTI0_IRQn);
  NVIC_ClearPendingIRQ(EXTI1_IRQn);

  //enabling
  NVIC_EnableIRQ(EXTI0_IRQn);
  NVIC_EnableIRQ(EXTI1_IRQn);

  // setting all array indexes to "&". (we consider this as a null index in array)
  resetArray();
}
```

تیکه کد زیر نیز مربوط به تابع، LCD_setCommand می باشد:

```
void LCD_setCommand(void){
  LCD_command(0x01);
  delay(10000);
  LCD_command(0x38);
  delay(100);
  LCD_put_char('S');
  delay(100);
  LCD_put_char('e');
  delay(100);
  LCD_put_char('i');
  delay(100);
  LCD_put_char('f');
  delay(100);
  LCD_put_char('i');
  delay(100);
  LCD_put_char('-');
  delay(100);
  LCD_put_char('m');
  delay(100);
  LCD_put_char('h');
  delay(100);
  LCD_put_char('m');
  delay(100);
  LCD_put_char('d');
  delay(100);
  LCD_put_char('p');
  delay(100);
  LCD_put_char('r');
  delay(100);
  LCD_command(0xC0);
  delay(100);
}
```

توضیح در رابطه با interrupt ها :

ما در حل این سوال همانطور که قبلا هم توضیح دادیم، فقط از دو تا از خطوط وقفه استفاده می کنیم که آنها را هم فقز برای پین های 0 و 1 پورت C ست می کنیم.
زیرا همانطور که در قسمت های قبل توضیح داده شد، برای اینکه اگر هر کدوم از ورودی های مربوط به keypad تغییر کند (از صفر به یک تبدیل شود) خروجی گیت OR ۴ ورودی ما هم که OR شده همین ۴ خروجی keypad است، یک میشود، و میفهمیم که یکی از ورودی ها عوض شده است. و دقیقا همین کار را هم برای کلید ها انجام دادیم و یک گیت OR ۲ ورودی هم قرار دادیم.
سپس در interrupt routine های مربوط به هرکدام از آنها، با توجه به کار های انجام شده، اینکه کدام کلید یا کدام یک از خانه های keypad فشار داده شده است را، پیدا و عملیات های مناسب را انجام می دهیم.

در تابع LCD_setCommand ابتدا میایم و دستور مربوط به پاک کردن کامل نمایشگر را انجام می دهیم، سپس نشانگر را به اولین خانه از خط اول می بریم و سپس کاراکتر ها را به ترتیب وارد می کنیم.

حال روتین هایی که در هنگام وقفه ها باید اجرا شوند را می نویسیم. اسم هر کدام از آنها مشخص و یکتاست. تابع EXTI0_IRQHandler که مربوط به خط صفر است و برای kepad است را بدین شکل می نویسیم:

```
//masking pending register
EXTI -> PR |= SET1(0);
//masking this intrrupt to avoid repeating this intrrupt being executed
EXTI -> IMR &= SET0(0);
//clearing pending IRQ
NVIC_ClearPendingIRQ(EXTI0_IRQn);
//finding kepad Button
findKeypadButton();
//for finding the number we changed them. now we reset them
GPIOB -> ODR |= SET1(A) | SET1(B) | SET1(C) | SET1(D);
if(status == 0){ // adding a character
  if(getNumber(array[size]) < 0){
    char newChar = array[size - 1];
    int operatorNumber = 0;
    for(int i = 1; i < size ; i++)
      if(getNumber(array[i]) < 0)
        operatorNumber++;
    delay(100);
    if(operatorNumber == 2){
      size--;
      delay(100);
      array[size] = '&';
      calculate();
      array[size++] = newChar;
      LCD_resetCommand();
    }else{
      LCD_put_char(array[size - 1]);
    }
  }else{
    LCD_put_char(array[size - 1]);
  }
}else if(status == 1){ // finding result
  calculate();
  LCD_resetCommand();
  status = 0;
}else if(status == 2){ // resetting the LCD
  resetArray();
  LCD_resetCommand();
  status = 0;
}
//unmasking this intrrupt
EXTI -> IMR |= SET1(0);
```

در این روتین ابتدا کار های مربوط به اجرا شدن روتین را که در ابتدای کد است (مانند MASK کردن pending register و همینطور MASK کردن interrupt mask register و همینطور پاک کردن interrupt pending request .)
سپس تابع findKeypadButton را صدا کردیم. کار این تابع این است که با توجه به ورودی های keypad و خروجی های آن، آن کلیدی که فشار داده ایم را پیدا می کند. سپس کار مورد نیاز را انجام می دهد، که اگر ورودی یکی از اعداد یا لگر های جمع و ... باشد، آن ها را به آرایه مان (قبلا توضیح دادیم) اضافه می کند و size را هم یکی اضافه می کند و یا اگر مساوی و یا clear بود، status را ست می کند.
سپس وقتی که به همین روتین باز می گردیم، با توجه به status میبینیم که در کدام وضعیت قرار داریم و کار مناسب با آن را انجام می دهیم.(البته اگر در حالتی باشیم که status = 0 باشد، آنگاه در صورتی که کلیدی که فشار داده شده، عملوندی مانند جمع یا .. باشد، میبینیم که آیا در رشته ای که الان در اختیار داریم، آیا یک عملوند دیگر (به جز خانه اول آرایه یا همان ایندکس صفر، زیرا ممکن است به خاطر منفی خود عدد ابشد، یعنی مثلا از قبل حاصلی منفی یک شده باشد، و این منفی برای آن باشد)؛ اگر داشتیم، پس ابتدا یک بار حاصل را حساب میکنیم و در ورودی مینویسیم و سپس این عملوند جدید را نشان می دهیم. دقیقا مطابق ماشین حساب ویندوز.)

حال تابع findKeypadButton را توضیح می دهیم. در این تابع، ابتدا با توجه به 4 تا if ای که قرار دادیم، ابتدا متوجه می شویم که کلید فشار داده شده در کدام ستون است. سپس با صفر کردن دونه دونه پین های ورودی keypad میبینیم که آیا ورودی میکرومان صفر شده است یا نه؛ اگر صفر شده بود، یعنی برای همان سطر است ولی اگر صفر نشده بود، یعنی برای آن سطر نیست. سپس با توجه به کلیدی که فشار داده شده است، عمل مناسب را انجام می دهیم.(به خاطر طولانی بودن کد، آن را اینجا نمیگذاریم.)

روتین EXTI1_IRQHandler برای این است که اگر کلیدی را فشار دادیم، متوجه آن شویم. سپس در این روتین میدانیم که قطعا کلیدی فشار داده شده است، سپس با دو تا if مشخص می کنیم که کدام کلید فشار داده شده است. سپس کار متناظر با آن را انجام می دهیم ولی ابتدا تابع calculate را صدا می کینیم تا در صورت نیاز حاصل عبارت موجود حساب شود و سپس تغییرات روی حاصل اعمال شود و سپس تابع inc_dec_number را صدا می کنیم.(با مقدار منفی یک در صورتی که کلید اول فشار داده شده باشد، و با مقدار یک در صورتی که کلید دوم فشار داده شده باشد.) تیکه کد زیر برای همین روتین می باشد:

```
void EXTI1_IRQHandler(void){ // buttons
  //masking pending register
  EXTI -> PR |= SET1(1);

  //masking this intrrupt to avoid repeating this intrrupt being executed
  EXTI -> IMR &= SET0(1);

  //clearing pending IRQ
  NVIC_ClearPendingIRQ(EXTI0_IRQn);

  if(GPIOB -> IDR & SET1(sw1)){
    calculate();
    inc_dec_number(-1);
    LCD_resetCommand();
  }else if(GPIOB -> IDR & SET1(sw2)){
    calculate();
    inc_dec_number(1);
    LCD_resetCommand();
  }

  //unmasking this intrrupt
  EXTI -> IMR |= SET1(1);
}
```

در انتهای هر بخش، آمدیم و متد LCD_resetCommand را صدا کردیم، در این تابع صرفا نشانگر LCD را به ابتدای خط دوم میبریم و سپس ۱۶ تا (به تعداد کل کاراکتر هایی که می توان در یک خط نشان داد) اسپیس مینویسیم در LCD ، سپس دوباره نشانگر را به ابتدای خط دوم میبریم و سپس تا جایی که آرایه پر باشد، عناصر آرایه را قرار می دهیم. تیکه کد مربوط به این تابع، به صورت زیر است:

```
void LCD_resetCommand(void){
  LCD_command(0xC0);
  delay(100);
  for(int i = 0; i < 16; i++){
    LCD_put_char(' ');
    delay(100);
  }
  LCD_command(0xC0);
  delay(100);
  int i = 0;
  while(i < size){
    LCD_put_char(array[i++]);
    delay(100);
  }
}
```

حال تابعcalculate را توضیح میدهیم. در این تابع ما ابتدا با توجه به کاراکتر های ورودی(آرایه گفته شده در ابتدا) عدد اول را از روی آرایه بدست می آوریم و در صورتی که کاراکتر اول آن '-' باشد، آن عدد را منفی میکنیم. سپس اگر دیگر آرایه تمام شده بود(فقط یک عدد داشتیم) خود آن عدد را در خروجی مینویسیم؛ وگرنه عملوند و عدد دوم را هم بدست میاوریم و سپس مقدار حاصل را در با توجه به عملوند بدست می آوریم و سپس آن را در خروجی می نویسیم.(چون در روشمان اگر از آرایه digits استفاده نمیکردیم، عدد از آخر به اول بود، پس از این آرایه استفاده کردیم و عدد را در آرایه جواب نوشتیم.) تیکه کد این قسمت به دلیل طولانی بودن آورده نشده است.

۳ تابع LCD_command و LCD_init و LCD_putChar با توجه به فایل کمکی و راهنمایی های آ« و همچنین دستورات مربوط به LCD نوشته شدند و به ترتیب مسئولیت دادن یک دستور جدید به LCD ، انجام تنظیمات اولیه آن مثل فعال کردن و آوردن نشانگر به اول خط و ... و قرار دادن یک کاراکتر جدید در LCD را داند. تیکه کد زیر مربوط به این ۳ تابع می باشد:

```
void LCD_put_char(char data) {
  GPIOA -> ODR = data;
  GPIOA -> ODR |= SET1(RS);
  GPIOA -> ODR &= SET0(RW);
  GPIOA -> ODR |= SET1(E);
  GPIOA -> ODR &= SET0(E);
}

void LCD_init(void){
  LCD_command(0x38);
  delay(100);
  LCD_command(0x06);
  delay(100);
  LCD_command(0x08);
  delay(100);
  LCD_command(0x0F);
  delay(100);
}

void LCD_command(unsigned char command){
  GPIOA -> ODR = command;
  GPIOA -> ODR &= SET0(RS);
  GPIOA -> ODR &= SET0(RW);
  GPIOA -> ODR |= SET1(E);
  delay(0);
  GPIOA -> ODR &= SET0(E);

  if (command < 4)
      delay(2);           /* command 1 and 2 needs up to 1.64ms */
  else
      delay(1);           /* all others 40 us */
}
```

تابع delay را هم دقیقا مطابق با همان فایل راهنما قرار دادیم، تنها به ورودی آن کلمه volatile را اضافه کردیم.

تابع getNumber صرفا یک کاراکتر را به عنوان ورودی میگیرد و در با استفاده از یک سوئیچ کیس، اگر آن کاراکتر، کاراکتر یک عدد بین 0 تا 9 باشد، همان عدد را برمیگرداند، در غیر این صورت، 1- را برذ میگرداند. تیکه کد زیر مربوط به این تابع می باشد:

```
int getNumber(char c){
  switch(c){
    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;
  }
  return -1;
}
```

تابع getChar هم یک جورایی معکوس تابع getNumber عمل میکند، به طوری که یک عدد به عنوان ورودی در اختیار میگیرد و سپس کاراکتر متناظر با آن را به عنوان خروجی برمیگرداند. تیکه کد زیر مربوط به ایت تابع است:

```
char getChar(int 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';
  }
  return '&';
}
```

تابع resetArray نیز صرفا مقدار size را صفر می کند و کل ایندکس های آرایه را هم '&' قرار میدهد.(این کاراکتر را نماد خالی بودن آرایه قرار داد میکنیم.) تیکه کد مربوط به این تابع به صورت زیر است:

```
void resetArray(void){
  size = 0;
  for(int i = 0; i< 16; i++)
    array[i] = '&';
}
```

حال به سراغ تابع inc_dec_number می رویم. این تابع می آید و با توجه به ورودی ای که در اختیار داریم، مقدار آن را با مقدار value که ورودی این تابع است جمع میکند و ابتدا آرایه و reset می کند با استفاده از تابع resetArray و سپس مقدار خروجی را همین مقدار جدید قرار می دهد. (نحوه خواندن ورودی و نوشتن ورودی درست همانند تابع calculate است.) تیکه کد مربوط به این تابع به صورت زیر است:

```
void inc_dec_number(int value){
  int number = 0;
  int counter = 0;
  if(getNumber(array[counter]) == -1){
    counter = 1;
    delay(100); // this is necessary, otherwise it won't work properly.(it seems, it doesn't have enough time to be executed)
  }
  while((counter < size) && (getNumber(array[counter]) >= 0)){
    number *= 10;
    number += getNumber(array[counter++]);
  }
  if(getNumber(array[0]) == -1){
    number = -number;
  }
  number += value;
  resetArray();
  if(number == 0){
    array[size++]='0';
    return;
  }
  char digit[8];
  if(number < 0){
    array[size++] = '-';
    number = -number;
  }
  int index = 0;
  while(number > 0){
    digit[index++] = getChar(number % 10);
    number /= 10;
  }
  while(index > 0)
    array[size++] = digit[--index];
}
```

منابع:

تحلیلی:

- https://www.electronicshub.org
- https://pediaa.com
- https://www.sciencedirect.com
- مطالب درس

کدی:

- https://digispark.ir/lcd-character-with-stm32-boards/
- https://digispark.ir/stm32-keypad/
- https://www.aparat.com/v/6ESNT?playlist=26086398
- https://embeddedcenter.wordpress.com/ece-study-centre/display-module/lcd-16x2-lm016l/
- https://www.keil.com/pack/doc/cmsis/DSP/html/index.html
- https://developer.arm.com/tools-and-software/embedded/cmsis
- https://stackoverflow.com/questions/60313895/stm32-gpio-interrupt-using-idr-register
- https://controllerstech.com/external-interrupt-using-registers/
- https://www.circuitstoday.com/matrix-keypad-interfacing-proteus
- https://circuitdigest.com/microcontroller-projects/keypad-interfacing-with-avr-atmega32
- https://microcontrollerslab.com/keypad-interfacing-8051-microcontroller/
- http://vlabs.iitkgp.ac.in/rtes/exp9/index.html#
- https://digispark.ir/stm32-keypad/
- https://www.bbc.co.uk/bitesize/guides/zd88jty/revision/5
- https://www.rapidtables.com/convert/number/binary-to-hex.html?x=01110000
- https://www.programiz.com/c-programming/examples/calculator-switch-case
- https://www.javatpoint.com/calculator-program-in-c
- https://www.includehelp.com/c-programs/calculator-using-switch.aspx
- https://www.tutorialspoint.com/how-to-write-a-simple-calculator-program-using-c-language
- https://blog.actorsfit.com/a?ID=00500-a0faba60-5eb5-4bae-9616-99ca497a76d5
- Datasheet
- Reference manual
- کلاس استاد و اسلاید ها