دانشگاه صنعتی شریف
دانشکده مهندسی کامپیوتر
پروژه درس ساختار و زبان کامپیوتر

عنوان:

پروژه اصلی سوم: ماشین‌حساب باینری با استفاده از آردوینو

Main Project 3: Binary Calculator Using Arduino

نگارش

گروه ۱:

سید احمد موسوی اول

سید امیرحسین موسوی فرد

رضا اسلامی ابیانه

آرمان طهماسبی زاده

استاد راهنما

دکتر حسین اسدی

بهمن ۱۴۰۳

فهرست مطالب

چکیده: در این سند خلاصه‌ای از اقدامات انجام شده در راستای پیاده‌سازی پروژه و نیز تمامی کدها و چالش‌های ایجاد شده در این فرایند همراه با توضیحات مربوطه قرار داده شده است.

واژه‌های کلیدی: آردوییِنو، wokwi، ماشین‌حساب، پایتون، رابط کاربری

۱ مقدمه

این پروژه طراحی و پیاده‌سازی یک ماشین‌حساب باینری است که توانایی دریافت اعداد به صورت باینری و اجرای چهار عمل اصلی را دارا باشد.
با استفاده از شبیه‌ساز wokwi و استفاده از رابط کاربری مناسب که با پایتون پیاده‌سازی شده است؛ این ماشین‌حساب قادر به نمایش معادلات ورودی و نتیجه محاسبات خواهد بود. در این سیستم ما خطاهای ورودی را به درستی شناسایی کرده و آن‌ها را با کدهایی یکتا مشخص کرده‌ایم.

۱-۱ تعریف مسئله

ماشین‌حساب پیاده‌سازی شده لازم است تا با دریافت ورودی به صورت باینری و همچنین یکی از چهار عملی اصلی و یا پرانتز باز یا پرانتز بسته، عبارت وارد شده از ابتدا را پردازش کرده و در صورت نامعتبر بودن این عبارت خطا گزارش کند. در غیر این صورت اما باید نتیجه پردازش شده عبارت را به صورت دهدهی در صفحه نمایش دهد. در این ماشین‌حساب لازم است تا ورودی‌های مختلف توسط یک رابط کاربری و دریافت ورودی‌های کاربر از دکمه‌های فشرده شده توسط او این اعمال را انجام دهد.

۱-۲ اهداف پژوهش

- فراهم کردن فرصت برای تجربه عملی با برد آردوینو، نمایشگر LCD و دکمه‌ها، و تقویت مهارت‌های عملی در زمینه الکترونیک و برنامه‌نویسی و آشنایی بیشتر با پیاده‌سازی عبارات ریاضی infix.
- طراحی و پیاده‌سازی یک رابط کاربری کارآمد که تجربه کاربری خوبی را فراهم کند و کاربران بتوانند به سادگی با ماشین‌حساب تعامل کنند.

- فراهم کردن فرصت برای تجربه عملی و کار با بردهای آردوینو و اجزای الکترونیکی مانند LED ها و دیپ‌سوئیچ‌ها.

۱-۳ موارد مورد استفاده

این پروژه با استفاده از آردوینو[۱] در شبیه‌ساز wokwi با استفاده از افزونه[۲] platform io پیاده‌سازی شده است. همچنین در این پروژه از زبان پایتون و کتابخانه‌های tkinter جهت طراحی رابط کاربری و serial جهت اتصال به شبیه‌ساز استفاده شده است.

۲ پیاده‌سازی

در این بخش به توضیح نحوه‌ی ساخت این پروژه شامل چالش‌های روبه‌رو شده، نحوه‌ی وصل کردن رابط کاربری با شبیه‌ساز و چگونگی کار با شبیه‌ساز به طور کامل توضیح میدهیم.

۱-۲ توضیحات ابتدایی

در ابتدا با توجه به شبیه‌سازهای معرفی شده تصمیم به کار با شبیه‌ساز wokwi گرفتیم؛ چرا که رابط کاربری ساده و قابل درکی داشت و می‌توانستیم با آن به راحتی کار کنیم.
می‌توانستیم به صورت برخط[۳] و داخل سایت wokwi نیز کارمان را انجام دهیم ولی به دلایل متعدد من جمله نحوه‌ی ارتباط برقرار کردن بین رابط کاربری و شبیه‌ساز باید پروژه را در سامانه‌ی محلی[۴] اجرا می‌کردیم؛ به همین دلیل از افزونه‌ی خود این سایت که مخصوص برنامه‌ی VS Code طراحی شده‌است؛ استفاده کردیم. همچنین از افزونه‌ی Platform IO نیز برای ساختن پروژه استفاده کردیم تا راحت‌تر بتوانیم از پروژه خروجی بگیریم.

---
[۱] arduino
[۲] extension
[۳] online
[۴] local system

۲-۲ مراحل پیاده‌سازی

برای اینکه کار ساده‌تر انجام شود؛ آن را به چند بخش کوچکتر تقسیم کردیم به طوری که در بخش اول ابتدا کد منطق ماشین‌حساب را با الگوریتم Shunting Yard که یکی از الگوریتم‌های معروف برای ساده‌سازی عبارات ریاضی به صورت infix است؛ پیاده‌سازی کردیم.
سپس در بخش دوم این کد زده شده که به زبان C زده شده‌بود را به زبان مخصوص آردوینو که همان زبان C++ تبدیل کردیم و پسوند فایل را ino. قرار دادیم. در ادامه ابتدا با استفاده از serial سعی کردیم که ورودی و خروجی مورد نظر را داشته باشیم؛ سپس در مرحله‌ی بعدی ابتدا سعی کردیم صرفا خروجی‌ها از serial باشند و به جای ورودی از دکمه‌هایی که به پین‌های شماره‌ی دو الی دوازده وصل کردیم استفاده کردیم.
در مرحله‌ی بعدی نیز از یک LCD برای نمایش خروجی درست و ورودی داده‌شده بهره گرفتیم و این انتهای راه شبیه‌سازی بود و در واقع پروژه آماده بود.
در آخر نیز از رابط کاربری که به زبان Python نوشتیم استفاده کردیم که در اینجا دوباره از ساز و کار serial استفاده کردیم و از این طریق دو برنامه را به یکدیگر وصل کردیم؛ در اصل به دلیل اینکه از شبیه‌ساز استفاده کردیم و این شبیه‌ساز در سامانه‌ی خودمان اجرا میشد لذا باید برای اینکه دو برنامه بتوانند از یکدیگر ورودی بگیرند و برای یکدیگر جواب ارسال کنند از port local با buad rate که مقدار آن را ۹۶۰۰ قرار دادیم استفاده کنیم؛ به این صورت رابط کاربری و شبیه‌ساز به یکدیگر متصل شدند و پروژه به پایان رسید.

۲-۳ چگونگی کار با شبیه‌ساز

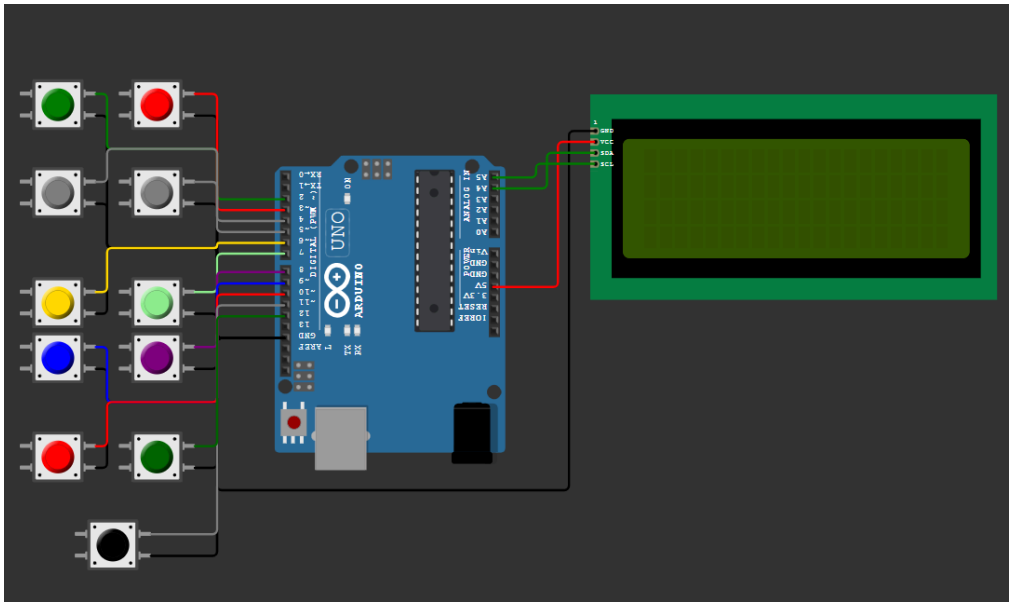
همانطور که اشاره شد از شبیه‌ساز wokwi استفاده کردیم؛ برای ادامه پروژه و برطرف کردن مشکلات کد؛ هر زمان که کد اصلی را تغییر میدادیم میتوانستیم با استفاده از افزونه Platform IO کد بدست آمده را دوباره کامپایل[۵] کنیم و فایل اجرایی با پسوند hex. بسازیم و دیگر نیازی نبود که کد را در برنامه‌ی آردوینو اجرا کنیم و از آنجا خروجی بگیریم و عملا تمام کارمان در برنامه‌ی VS Code به سادگی انجام میشد چرا که کافی بود با استفاده از Build که افزونه‌ی Platform IO در اختیارمان میگذاشت برنامه را کامپایل کرده و سپس شبیه‌ساز را با استفاده از افزونه‌ی wokwi اجرا کنیم و خروجی را ببینیم.
در این پروژه یک فایل به نام diagram.json وجود داشت که در آن میتوانستیم قسمت بصری پروژه را

[۵] compile

تغییر داده و اتصالات و قطعات را در آنجا ببینیم و تغییرات لازم را بدهیم که عکس‌های زیر بخشی از کد آن به علاوه‌ی شبیه‌ساز همان فایل را نمایش میدهد.

```
"connections": [
  [ "lcd1:GND", "uno:GND.1", "black", [ "h-22", "v313.2", "h-307.2", "v-132.7" ] ],
  [ "lcd1:VCC", "uno:5V", "red", [ "h-31.6", "v126.5" ] ],
  [ "lcd1:SDA", "uno:A4", "green", [ "h-41.2", "v31" ] ],
  [ "lcd1:SCL", "uno:A5", "green", [ "h-50.8", "v12" ] ],
  [ "btn2:2.r", "uno:GND.1", "black", [ "h19.4", "v193.9" ] ],
  [ "btn4:2.r", "uno:GND.1", "black", [ "h19.4", "v115.4" ] ],
  [ "btn6:2.r", "uno:GND.1", "black", [ "h19.4", "v19.4" ] ],
  [ "btn7:2.r", "uno:GND.1", "black", [ "h19.4", "v-28.6" ] ],
  [ "btn11:2.r", "uno:GND.1", "black", [ "h19.4", "v-115" ] ],
  [ "btn10:2.r", "uno:GND.1", "black", [ "h57.8", "v-191.8" ] ],
  [ "btn3:2.r", "uno:GND.1", "black", [ "h9.8", "v38.6", "h96", "v76.8" ] ],
  [ "btn5:2.r", "uno:GND.1", "black", [ "h9.8", "v-57.4", "h96", "v76.8" ] ],
  [ "btn1:2.r", "uno:GND.1", "black", [ "h9.8", "v115.4", "h96", "v76.8" ] ],
  [ "btn8:2.r", "uno:GND.1", "black", [ "h9.8", "v29", "h96", "v-57.6" ] ],
  [ "btn9:2.r", "uno:GND.1", "black", [ "h9.8", "v-57.4", "h96", "v-57.6" ] ],
  [ "btn1:1.r", "uno:2", "green", [ "v0", "h9.8", "v48", "h96", "v48" ] ],
  [ "btn2:1.r", "uno:3", "red", [ "v0", "h19.4", "v105.6" ] ],
  [ "btn3:1.r", "uno:4", "grey", [ "v0", "h9.8", "v-28.8", "h96", "v67.2" ] ],
  [ "btn4:1.r", "uno:5", "grey", [ "v0", "h19.4", "v48" ] ],
  [ "btn5:1.r", "uno:6", "gold", [ "v0", "h9.8", "v-38.4", "h96", "v-9.6" ] ],
  [ "btn6:1.r", "uno:7", "lightgreen", [ "v0", "h19.4", "v-28.8" ] ],
  [ "btn7:1.r", "uno:8", "purple", [ "v0", "h19.4", "v-67.2" ] ],
  [ "btn8:1.r", "uno:9", "blue", [ "v0", "h9.8", "v48", "h96", "v-105.6" ] ],
  [ "btn9:1.r", "uno:10", "red", [ "v0", "h9.8", "v-38.4", "h96", "v-96" ] ],
  [ "btn10:1.r", "uno:11", "grey", [ "v0", "h57.8", "v-201.6" ] ],
  [ "btn11:1.r", "uno:12", "darkgreen", [ "v0", "h19.4", "v-115.2" ] ]
],
```

شکل ۱: کد بخش اتصالات



شکل ۲: شبیه سازی همان کد

۲-۴ بررسی منطق کد

همانطور که اشاره شد کدها در فایلی با پسوند ino. نوشته شدند؛ در این فایل ما برای سادگی در اجرای کد و فهم بیشتر آن را به توابعی مختلف شکاندیم تا کار هر بخش توسط یک تابع انجام گیرد که در اینجا ابتدا در عکس زیر میتوانید عکس پروتوتایپ[۶] توابع استفاده شده را ببینید که در ادامه توضیحات آن نیز برای هر تابع داده میشود.

```
// Function prototypes
void pushNumber(IntStack *stack, long int value);
long int popNumber(IntStack *stack);
void pushChar(CharStack *stack, char value);
char popChar(CharStack *stack);
int isOperator(char c);
int precedence(char op);
long int applyOperation(long int a, long int b, char op);
long int shuntingYardExpression(const char *expression);
char* convertBinaryInputToDecimalInput(char* input);
int haveErrorParantesesOrDividing(char* expression);
int weHaveErrorInExpression(char* expression);
long int writeOnLCD();
void writeOnLCD(String result);
void writeExpression(String decimal);
void upadateExpressionAfterEqual();
int readButton();
void letsExit();
```

شکل ۳: پروتوتایپ توابع

۲-۴-۱ تابع setup آردوینو

در این تابع ابتدا روی 9600 buad port سریال آردوینو را باز کرده تا ارتباط بین ماشین حساب و رابط کاربری از این طریق برقرار شود. همچنین LCD را راه‌اندازی کرده و کلیدهای وارد شده را نیز در حالت آماده به دریافت ورودی از کاربر قرار می‌دهد.

۲-۴-۲ تابع loop آردوینو

این تابع ابتدا با استفاده از تابع readButton که در ادامه توضیح داده شده است دریافت ورودی را چک می‌کند و با استفاده از عبارات شرطی بر حسب کاراکتر ورودی تابع مورد نظر را فراخوانی می‌کند. همچنین به دلیل اتصال به رابط کاربری در هر بار فراخوانی بررسی می‌شود که آیا روی سریال مقداری نوشته شده است یا خیر.

---

[۶] Prototypes

۲-۴-۳ توابع push , pop: number, char

همانطور که گفتیم از الگوریتم Shunting Yard استفاده کردیم که در این الگوریتم نیاز به دو پشته یکی مخصوص عملیات‌ها و یکی مخصوص اعداد داریم و برای راحتی استفاده از پشته‌ها این توابع را نوشتیم.

۲-۴-۴ تابع isOperator

این تابع بررسی می‌کند که آیا کاراکتر وارد شده یکی از چهار عمل اصلی هست یا خیر.

۲-۴-۵ تابع precedence

این تابع به نسبت اولویت انجام عملیات ریاضی مقدار ۱ یا ۲ را باز می‌گرداند. در صورتی که عملیات یکی از جمع یا تفریق بود مقدار ۱ بازگردانده شده و در غیر این صورت مقدار ۲ بازگردانده می‌شود.

۲-۴-۶ تابع applyOperation

در این تابع با ورودی گرفتن دو عدد و یک عملیات نتیجه اعمال عملیات وارد شده روی دو عدد را باز می‌گرداند. همچنین در این تابع در صورتی که تقسیم بر صفر وارد شده بود، جهت جلوگیری از خطاهای احتمالی مقدار ___LONG_MAX___ بازگردانده می‌شود.

۲-۴-۷ تابع shuntingYardExpression

این تابع که تابع اصلی می‌باشد با ورودی گرفتن یک عبارت به صورت آرایه‌ای از کاراکترها الگوریتم shunting yard را روی آن اجرا کرده و نتیجه را باز می‌گرداند.

۲-۴-۸ تابع convertBinaryInputToDecimalInput

مطابق نام‌گذاری انجام شده این تابع ورودی باینری کاربر را به ورودی ده‌دهی تبدیل می‌کند. در واقع تنها اعداد باینری به اعداد ده‌دهی تبدیل می‌شوند و این تابع تغییری روی باقی رشته و جایگاه عملیات‌ها نمی‌دهد.

۹-۴-۲ توابع بررسی خطای احتمالی رشته و به دست آوردن کد خطا

این مجموعه توابع با ورودی گرفتن یک رشته همه خطاهای موجود در رشته را یافته و برای هر خطا یک کد خطا که عددی از ۱ تا ۵ می‌باشد را خروجی می‌دهد. در صورتی که خطایی وجود نداشت نیز مقدار صفر بازگردانده می‌شود. هر یک از کد خطاهای بازگردانده شده معادل خطای زیر می‌باشد:

- کد ۱: تقسیم بر صفر
- کد ۲: کمتر بودن تعداد پرانتزهای باز شده نسبت به پرانتزهای بسته شده
- کد ۳: کمتر بودن تعداد پرانتزهای بسته شده نسبت به پرانتزهای باز شده
- کد ۴: عدم وجود یک عبارت عددی بین دو عملیات
- کد ۵: آخرین کاراکتر وارد شده یک عملیات است و عبارت تکمیل نشده

```
int weHaveErrorInExpression(char* expression) {
    /**
     * This function is for finding All errors in expression
     *
     * This function will return:
     *      1. dividing by zero
     *      2. we have unmatched paranteses
     *      3. we have open paranteses
     *      4. if we have two operator that are left to each other we have error
     *      5. if last char is not digit we can't calculate
     *      0. NO Error!
     **/

    int len = strlen(expression);
    // NO Error in len = 0:
    if (len == 0) return 0;

    // if we have two operator that are left to each other we have error:
    for (int i = 0; i < len - 1; i++) {
        if (isOperator(expression[i]) && isOperator(expression[i + 1])) return 4;
    }

    // if last char is not digit we can't calculate:
    if (expression[len - 1] != '1' && expression[len - 1] != '0' && expression[len - 1] != ')') return 5;

    return haveErrorParantesesOrDividing(expression);
}
```

شکل ۴: انواع خطاهای موجود

۱۰-۴-۲ توابع نوشتن روی LCD

در این تابع تمامی منطق نوشته شدن روی LCD پیاده‌سازی شده است که در خط اول ورودی کاربر به صورت باینری نمایش داده شده، در خط دوم عبارت پردازش شده به صورت ده‌دهی نمایش داده می‌شود و در خط سوم نیز مقدار نهایی عبارت، در صورت عدم وجود خطا، نمایش داده می‌شود.

۱۱-۴-۲ تابع updateExpressionAfterEqual

پس از فشرده شدن دکمه مساوی مقدار عبارت حساب شده و نمایش داده می‌شود. علاوه بر این، مقدار باینری عبارت محاسبه شده نیز به جای عبارتی که کاربر تا اینجا وارد کرده بود جایگزین می‌شود.

۱۲-۴-۲ تابع readButton

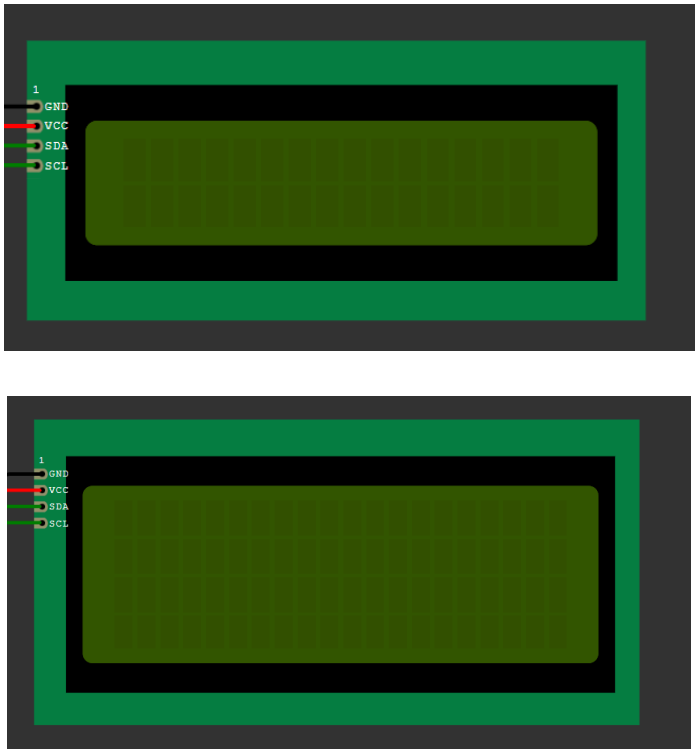
این تابع در تابع loop فراخوانی شده و با هربار فراخوانی بررسی می‌کند که آیا یکی از دکمه‌ها فشرده شده است یا خیر. همچنین در صورتی که دکمه‌ای فشرده شده بود این تابع کاراکتر مناسب مربوطه را باز می‌گرداند.

۱۳-۴-۲ تابع letsExit

این تابع که هنگام فشرده شدن دکمه خروج فراخوانی می‌شود عملیات خروج را با خاموش کردن LCD و وارد شدن به یک لوپ بینهایت انجام می‌دهد.

۵-۲ چالش‌های پیاده‌سازی

یکی از اولین چالش‌هایی که در این پروژه با آن برخورد کردیم عدم آشنایی با شبیه‌ساز wokwi بود که آشنایی با این شبیه‌ساز و راه‌اندازی محیط کار مربوطه در vscode با استفاده از راهنمایی‌های وبسایت شبیه‌ساز wokwi انجام شد.
چالش بعدی این بود که در ابتدا الگوریتم shunting yard را با استفاده از زبان C پیاده‌سازی کرده بودیم که ورودی را از ترمینال دریافت می‌کرد. اما برای اتصال کد مربوطه به شبیه‌ساز و برد آردوینو باید این الگوریتم را به زبان C++ تبدیل می‌کرده و همچنین ورودی و خروجی را با استفاده از سریال دریافت می‌کردیم.
یکی دیگر از مشکلاتی که داشتیم این بود که رشته‌های نمایش داده شده روی LCD جا نمی‌گرفتند و اندازه LCD اولیه که ۱۶ در ۲ بود برای این استفاده کافی نبود. در نهایت پس از گشت و گذار بین LCD های موجود بزرگترین LCD ای که پیدا کردیم ۲۰ در ۴ بود که همچنان هم کوچک است اما نتیجه بهتری نسبت به LCD قبلی دریافت کردیم.



شکل ۵: تفاوتLCD های مختلف

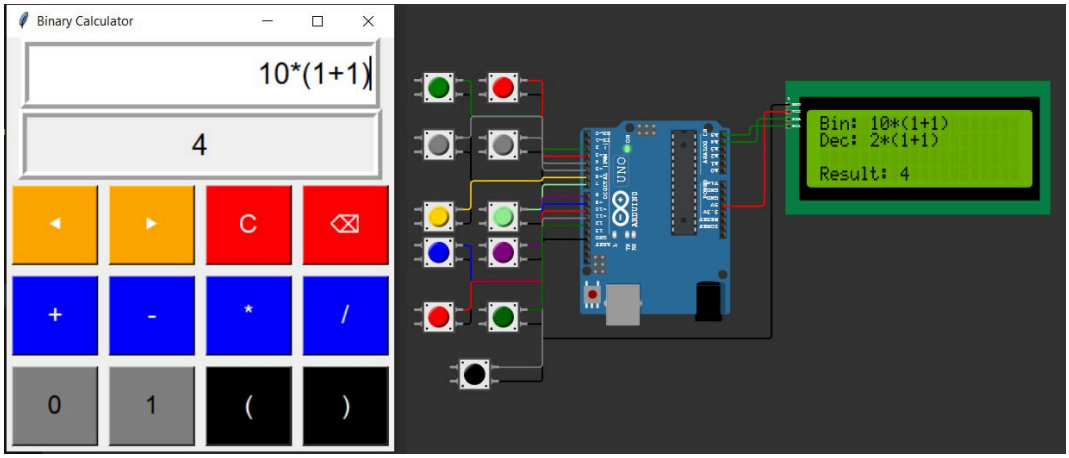
در ادامه لازم بود تا ورودی‌هایی که به صورت سریال بودند به ورودی‌های مربوطه که با فشردن دکمه‌ها در شبیه‌ساز انجام می‌گرفتند تبدیل می‌شدند. همچنین لازم بود تا خروجی‌های لازمه نیز که با استفاده از سریال انجام می‌گرفتند را هم به خروجی قابل نمایش روی LCD تبدیل می‌کردیم.
چالش بعدی نحوه جایگذاری دکمه‌ها و قطعات در شبیه‌ساز wokwi بود که باید طوری این جایگذاری را انجام می‌دادیم تا قطعات روی هم قرار نگیرند و همچنین سیم‌کشی‌ها نیز مشخص و قابل دنبال کردن باشند. در اینجا یکی از چالش‌های دیگر این بود که در افزونه wokwi که در محیط vscode داشتیم قابلیت نمایش سیم‌کشی‌ها و قابلیت ادیت همزمان آن‌ها را در نسخه مجانی نداشت و مجبور به استفاده از وبسایت شبیه‌ساز شدیم که در این وبسایت این قابلیت در دسترس باشد.
در مرحله بعدی لازم بود تا یک رابط کاربری با استفاده از زبان پایتون پیاده‌سازی شود که چالش‌های مربوط به خودش را داشت. پس از یافتن کتابخانه‌های مورد نیاز در این پیاده‌سازی و یاد گرفتن کار با آن‌ها رابط کاربری را پیاده‌سازی کردیم اما به چالش‌های مختلفی برخوردیم. یکی از این چالش‌ها عدم هماهنگی بین کد پایتون و شبیه‌سازی آردوینو بود که سرعت ورودی گرفتن پایتون و شبیه‌ساز و همچنین نوع ارتباط آن‌ها که مختلف بود باعث ایجاد خطاهای مختلفی می‌شد. مشکل دیگر این

بود که کد پایتون همزمان با محاسبه عبارت توسط ماشین‌حساب تلاش در بروزرسانی عبارت وارده و رابط کاربری بود که باعث ایجاد خطا می‌شد. در نهایت با استفاده از delay در آردوینو و time در پایتون هماهنگ‌سازی‌های لازمه را پیاده‌سازی کردیم و خطاها رفع شد.

۳ جمع‌بندی و خروجی نهایی

در نهایت پس از رفع چالش‌های مطرح شده و دیگر چالش‌های جزئی که از بیان کردن آن‌ها صرف‌نظر شده است پیاده‌سازی کدها تکمیل شده و خروجی حاصل شد.

در این ماشین حساب با وارد کردن ورودی‌ها به رابط کاربری در پایتون، خروجی مربوطه در صفحه نمایش شبیه‌ساز و همچنین صفحه نمایش رابط کاربری پایتون بروزرسانی شده و نتیجه یا خطای حاصل شده نمایش داده می شود.



شکل ۶: خروجی نهایی