**پیش­گزارش دستورکار دوم آزمایشگاه ریزپردازنده و زبان اسمبلی**

نگار موقتیان، 9831062

1. **کدهای مورد نیاز برای برنامه ریزی برد:**

فایل مربوط به کدهای نوشته شدۀ برنامه به پیوست ارسال می­گردد.

ابتدا تنظیمات مربوطه به keypad انجام شده­اند (در این آزمایش از یک keypad 4x4 استفاده شده است). در تابع setup سریال راه اندازه شده و پین­های خروجی برد مشخص شده­اند. به علاوه برای سهولت درگرفتن ورودی از virtual terminal میزان timeout آن به 250 میلی ثانیه کاهش داده شده است.

تابع setLED برای روشن کردن LED ها مطابق خواستۀ دستور کار نوشته شده است. این تابع به عنوان ورودی عددی دریافت می­کند که نشانگر تعداد LED هایی که باید روشن شوند است. ابتدا این عدد بررسی می­شود تا مطمئن شویم عددی بین 0 تا 9 است و در غیر این صورت پیغام خطای مناسب را چاپ کنیم. سپس به ترتیب از سمت چپ LED هایی که باید روشن باشند را روشن کرده و باقی را خاموش می­کند.

در تابع loop نیز بررسی می­کنیم که عددی در keypad و یا virtual terminal وارد شده است یا خیر. در صورتی که عددی وارد شده بود ابتدا مقدار آن در ترمینال چاپ شده و سپس تابع setLED صدا زده می­شود (عدد ورودی به عنوان آرگومان به این تابع پاس داده می­شود).

1. **انواع keypad ماتریسی و چگونگی کارکرد آن­ها:**

یک نوع ساده از این keypad ها keypad های عادی هستند که صفحۀ آن­ها از تعدادی میکروسوئیچ به ازای هر دکمه تشکیل شده. با فشار دادن هر دکمه و اتصال میکروسوئیچ متناظر با آن می­توان فشرده شدن دکمه را تشخیص داد. در این keypad های ماتریسی هر سطر و هر ستون یک پین متناظر دارد. پایه­های R1 تا R4 به پایه­های خروجی میکروکنترلر متصل می­شوند و در حالت عادی ولتاژ HIGH دارند. پایه­های C1 تا C3 نیز به صورت ورودی و در حالت pull-up در نظر گرفته می­شوند. به صورت نرم­افزاری و در توابع از پیش تعریف شده در کتابخانۀ Keypad مقدار پایه­های R به ترتیب LOW می­شوند و به ازای هر یک بررسی می­شود که آیا هیچ یک از پایه­های C نیز LOW شدند یا خیر. اگر برای مثال پایۀ Ri را LOW کرده بودیم و دیدیم که پایۀ Cj هم LOW شده است، این بدین معناست که دکمه موجود در ردیف i و ستون j فشرده شده­است (با این روش فشرده شدن همزمان چند دکمه نیز قابل تشخیص است).

شکل 1 – نمونه­ای از یک keypad ماتریسی عادی

نوع دیگر keypad ها keypad های خازنی هستند که در آن­ها صفحات به خازن­های شارژ شده­ای متصل هستند. با تماس دست با آن­ها این خازن­ها تخلیه شده و سیستمی که در این keypad ها تعبیه شده تشخیص می­دهد که کدام دکمه لمس شده است.

1. **پدیدۀ نوسان (bounce) کلید چیست و چگونه می توان از بروز اشکالات ناشی از آن جلوگیری کرد؟**

فرض کنید یکی از پایه های ورودی به یک دکمه فشاری متصل است (یا یک دکمه از keypad). ما یک بار دکمه را فشار می دهیم اما در واقعیت در خروجی این دکمه یک انتقال تمیز از صفر به یک نداریم. به دلیل این که کلید فیزیکی است ممکن است چندین بار اتصال قطع و وصل شود تا بالاخره ثابت بماند. این قضیه باعث می­شود پردازنده به جای یک بار فشرده شدن دکمه چندین تغییر را حس کند و برنامه کارکرد مورد انتظار را نداشته باشد.

برای حل این مشکل می­توان از debouncing استفاده کرد. Debouncing باعث می شود تغییر دکمه تنها یکبار به پردازنده خبر داده شود (این کار با بررسی عرض پالس انجام می­شود). در میکروکنترلری که با آن کار می­کنیم می­توان این قابلیت را بر روی پایۀ ورودی فعال کرد تا از بروز این مشکل جلوگیری کنیم. به علاوه می­توان از یک خازن یا یک latch برای جلوگیری از این پدیده استفاده کرد.

****

شکل 2 – سیگنالی به شکل پایین می­تواند توسط debouncing به سیگنال بالا تبدیل شود.

1. **تعریف مختصر توابع مورد نیاز از کتابخانه Keypad.h:**

* Keypad(makeKeymap(userKeymap), row[], col[], rows, cols)

این تابع در حقیقت constructor کلاس Keypad برای ایجاد یک شیء جدید از این کلاس می­باشد. آرگومان اول آن آرایۀ کاراکترهای موجود بر روی keypad می­باشد که خود به تابع makeKeymap پاس داده شده تا به فرمت مناسب تبدیل شود. آرگومان بعدی آن آرایه­ای از شماره پین­های متصل شده به سطرهای keypad و آرگومان بعدی نیز آرایه­ای از شماره پین­های متصل شده به ستون­های keypad می­باشد. پس از آن نیز باید تعداد سطرها و ستون­های keypad را مشخص نماییم.

* char getKey()

دکمه­ای که فشرده شده است را در صورت وجود برمی­گرداند. این تابع non-blocking است، بدین معنا که تنها بررسی می­کند که دکمه­ای فشرده شده یا خیر و منتظر فشرده شدن دکمه­ای نمی­شود (بدین ترتیب ادامۀ کد می­تواند در همان لحظه اجرا شود و اجرای برنامه متوقف نمی­شود).

* bool getKeys()

مشخص می­کند که آیا تعدادی دکمه فشرده شده­اند یا خیر. این تابع تنها یک مقدار boolean بر می­گرداند و برای مشخص کردن این که کدام دکمه­ها فشرده شده­اند باید از تابع دیگری استفاده کنیم.

* char waitForKey()

مانند getKey مقدار دکمۀ فشرده شده را برمی­گرداند، با این تفاوت که blocking است تا ابد منتظر می­ماند تا کاربر دکمه­ای را فشار دهد. زمانی که این تابع منتظر ورودی کاربر است هیچ یک از کدهای موجود پس از این تابع اجرا نمی­شوند.

* KeyState getState()

حالت هر یک از دکمه­ها را برمی­گرداند. این حالات می­توانند یکی از مقادیر IDLE (کلید فشرده نشده)، PRESSED (کلید در این لحظه فشرده شده)، RELEASED (کلید در این لحظه رها شده) و HOLD (کلید فشرده نگه داشته شده - معیار فشرده نگه داشته شدن کلید با استفاده از تابع setHoldTime مشخص می­شود) را به خود بگیرند.

* boolean keyStateChanged()

اگر حالت هر یک از دکمه­های Keypad تغییر کند می­توانیم این تغییر را با استفاده از خروجی این تابع متوجه شویم.

1. **نحوه و کاربردهای ارتباط سریال در آردوینو:**

ارتباط سریال برای ارتباط میان دو دستگاه مانند دو میکروکنترلر یا یک میکروکنترلر و کامپیوتر از طریق یک پروتکل مشخص استفاده می­شود. برای یک ارتباط دو طرفۀ سریال حداقل به 3 سیم نیاز داریم. یک سیم GND که ولتاژ مرجع میان دو برد را تعیین می­کند و دو سیم دیگر به نام سیم­های RX و TX که یکی مربوط به خروجی و دیگری مربوط به ورودی می­شود (R از کلمۀ Receive و T از کلمۀ Transfer می­آید). در واقع دو دستگاهی که از طریق ارتباط سریال به یکدیگر متصل می­شوند از طریق یکی از این سیم­ها داده را ارسال کرده و از طریق دیگری داده را دریافت می­کنند، لذا پین RX یک برد باید به TX دیگری و پین TX آن باید به RX دیگری متصل شود.

برد آردوینویی که از آن استفاده می­کنیم از 4 ارتباط سریال متناظر با پایه­های TX0 و RX0 (که پایه­های 0 و 1 برد هستند) تا TX3 و RX3 پشتیبانی می­کند. در برنامه­­ای که می­نویسیم این سریال­ها با Serial، Serial1، Sertial2 و Serial3 تعریف شده­اند. علاوه بر این پایه­ها می­توان از دیگر پین­ها برای ارتباط سریال استفاده کرد اما در این صورت مدیریت آن­ها به صورت نرم­افزاری و توسط کتابخانۀ SoftwareSerial انجام می­شود. هر یک از این Serial ها می­تواند داده­ای را بخواند یا بنویسد.

1. **تعریف مختصر و نحوه کار با توابع ارتباط سریال:**

* begin()

این تابع اتصال سریال را برقرار می­کند و از آن لحظه سریال آمادۀ ارسال و دریافت اطلاعات می­شود. این تابع به عنوان آرگومان سرعت ارتباط سریال را بر حسب بیت بر ثانیه دریافت می­کند. همچنین یک آرگومان optional به نام config دارد که برای یکسری از تنظیمات به کار می­رود (برای مثال برای تنظیم بیت parity و ...).

* end()

بر عکس تابع قبل ارتباط سریال را می­بندد و پین­های RX و TX را برای دیگر استفاده­ها آزاد می­کند.

* find()

داده­ها را از بافر سریال می­خواند تا زمانی که به داده­ای مشخص (که به عنوان آرگومان به این تابع داده می­شود) برسد. در صورت وجود چنین داده­ای true و در غیر این صورت false برمی­گرداند.

* parseInt()

به دنبال اولین عدد معتبر در ورودی سریال می­گردد. اگر در زمان مشخصی (پیش از زمان timeout) چنین عددی را دریافت کرد آن را برگردانده و در غیر این صورت مقدار صفر برمی­گرداند. به علاوه آرگومان­هایی optional دارد که یکی برای مشخص کردن عملکرد برنامه زمانی که ورودی مخلوطی از اعداد و دیگر کاراکترها باشد و یکی برای کاراکترهایی که می­توانند زمان parse کردن نادیده گرفته شوند به کار می­رود.

* println()

داده­ای را به صورت متن قابل فهم برای انسان به خروجی سریال می­فرستد (با استفاده از کاراکترهای ASCII)، در انتها نیز به خط بعد می­رود. به علاوه می­توان یک فرمت نیز برای آن تعیین کرد، برای مثال مشخص کنیم که خروجی به فرمت باینری باشد، یا هگزادسیمال یا... .

* read()

اولین بایت از ورودی سریال را خوانده و آن را برمی­گرداند.

* readStringUntil()

ورودی سریال را تا زمان دریافت یک کاراکتر خاص (که به عنوان ورودی به آن داده می­شود) و یا سپری شدن timeout دریافت کرده و رشته دریافت شده را به صورت یک String برمی­گرداند.

* write()

داده­ای را به صورت رشته بیت در خروجی سریال می­نویسد. به عنوان ورودی می­تواند یک مقدار خاص، یک String و یا یک آرایه به صورت بافر (به همراه طول آن) را دریافت کرده، آن را به یک یا تعدادی بایت­ تبدیل کرده و در خروجی بنویسد.