情報科学実験Ｂ　課題2レポート

氏名　山久保孝亮
所属　大阪大学基礎工学部情報科学科ソフトウェア科学コース
メールアドレス　u327468b@ecs.osaka-u.ac.jp
学籍番号　09B22084

提出日　2024 年 5 月 20 日
担当教員　小南大智, 松本麻佑

1　システムの仕様

今回の課題で作成した私の電卓の仕様は以下の通りである.

- 二つの数値の四則演算を実行する. 即ち,1+2 のような計算を実行する. 今回の課題では 1+1+1 のような 3 つ以上の数値の計算には対応していない.
- 初期状態では 7 セグメントデコーダは全消灯している. また, 今回の電卓では回路上で HIGH か LOW かを決める 4 つの入力があり, その 4 つの入力を操作してからスイッチを開放したときにのみ出力が変化する.
- 0 から 9 が入力されるとそのままの数値が,10 から 13 が入力されると小さい順に+,-,*,/が入力されたと考える.15 を入力しスイッチを開放すると結果を計算し, もう一度スイッチを開放すると結果を表示する.
- 入力する際の二つの数値は複数桁にも対応している. 例えば 1 と 2 を連続で入力した際には 12 という数値を表すようにした. また, 桁数は最大 3 桁で 4 桁以上の数値が入力すなわち 4 回以上連続で数値が入力された場合はエラーとする. また,01 のように数値を入力した際は 1 を数値として考えるが,0001 のように 4 つ以上の数値が入力された場合は前述の仕様により, エラーが表示される.
- 答えが複数桁の際にはスイッチ解放の度に上位桁から順番に表示する. 全ての桁を出力してから再びスイッチを開放したときに初期状態に戻る.
- エラー処理は 7 セグメントデコーダに入力が 14 のときの出力を 1 秒間表示させた後に初期状態に戻る.
- 引き算において計算結果が負の数になった場合は絶対値となって答えを出力する. 即ち,1-4 を実行すると-3 ではなく 3 を出力するという仕様である. また, 割り算において 0 で割った際はエラーとなる.
- 数値, 演算子, 数値の形に従わない数式が入力された場合はエラーが表示される.

2　ハードウェア回路

今回の電卓の課題ではプッシュスイッチの制御回路,7 セグメント LED の制御回路,Arduino の入力と出力をつなぐ回路を実現した. 以下でこれらの詳細について順に述べる.

1. プッシュスイッチの制御回路は以下の図 1 の指導書にあったチャタリング防止付きの回路を実装した.[1] この回路は以下の図 2 のように, スイッチが上のような挙動ではなく下のような挙動になってしまうことを防ぐものである.[2]

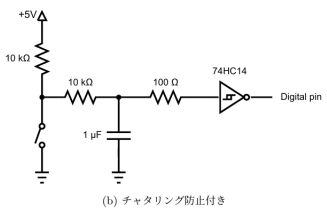


図 1: チャタリング防止回路

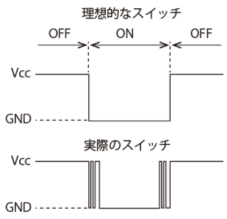


図 2: スイッチの様子

チャタリング防止回路はコンデンサにたまった電荷が急に変化しないという性質を利用して, チャタリングによってスイッチのオンオフが繰り返し行われたとしても, 実際の電圧の変化はコンデンサの

電荷の減少に従うようにして影響を受けないようにしたものである. コンデンサの左側二つの抵抗はこの電荷の減少の速さを抑えて影響を受けない時間を長くするという目的のため置かれている. ただし, コンデンサによって電圧が HIGH と LOW の中間の値をとってしまうので, その中間地を HIGh と LOW に再び分類するためにシュミットトリガ・NOT が置かれている. その直前の抵抗は, シュミットトリガ・NOT の電源電圧の最大値が 6V であることから, 電圧の値を下げるということが目的であると考えられる.[3] そして図 1 の Digital pin を Arduino の指定したピンにつなぐ.(私の場合はピン 8)

2. 7 セグメント LED の制御は以下の図 3 の 7447 デコーダのアルファベットに対応する場所を図 4 の LED の対応するピンにつなぐことで実現した.[1]

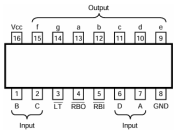


図 3: 7447 デコーダ

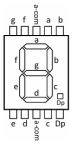
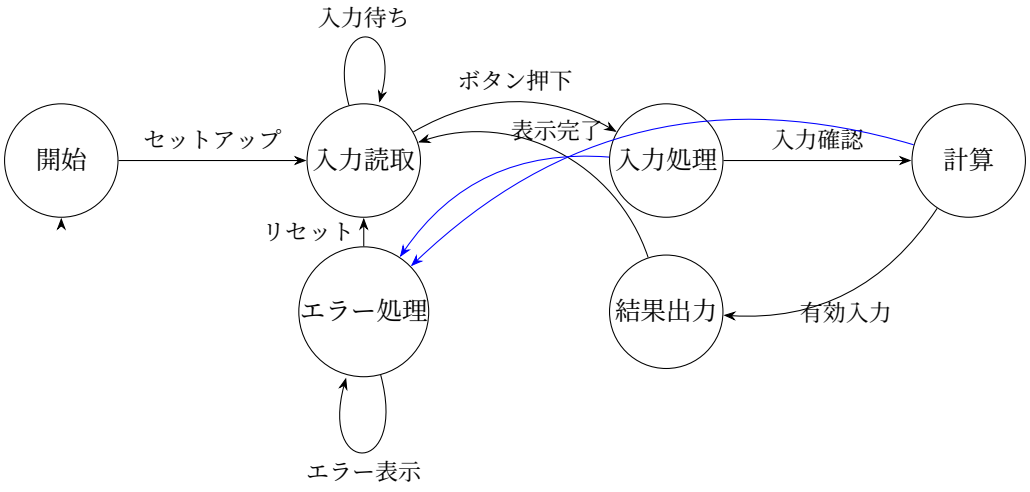


図 4: 7 セグメント LED

3. 7447 デコーダの A から D に対応するピンには,arduino 内部で処理した後の出力を与えるので Arduino 指定したピンとつないだ.(私の場合はピン 4 ~ピン 7)arduino への入力は 5V またはグラウンドにつないだ線を arduino の指定したピンに差し込んだ.(私の場合はピン 9 ~ピン 12)

3 制御プログラム

私が作成した電卓の制御プログラムの状態遷移図は以下のようになる.



青色の有効辺はエラー処理に向かう有効辺である. エラー処理, 入力処理, 計算, 結果出力はそれぞれ関数として表現し, デバッグを行いやすくした. また, 状態遷移はスイッチが解放された瞬間のみに行われるという機能を実装するために change という変数を用意し, スイッチが解放されたときにのみ change が 1 に, それ以外では 0 という条件分岐を実現することで非トラッキング回路で実現した. また, 数式の記憶は要素数 7 個の配列 memory を, 表示する計算結果の記憶は要素数 6 個の配列 answer_memory をそれぞれグローバル変数として使用した. 以下では, 上記の関数と loop 関数の実装について記述する.

3.1 loop 関数

loop 関数ではまずエラーが起きたかどうか (error_flag) を判定する.error_flag が 1 であれば後述の error 関数を呼び出して終了する.1 でなければ, スイッチを開放したかどうか (change) のフラグを判定する. スイッチが解放された瞬間を検出するために, 直前にスイッチが押されていたかどうかを記憶する変数 previousbuttonState と現在のスイッチの状態を表す buttonState を使用する. 直前は押していて, 現在は押していない即ち previousbuttonState が 1 で buttonState が 0 となった際に change を 1 に変更する. change が 1 でなければ,loop 関数を終了する.change が 1 であれば,change を 0 に戻して結果を出力するかどうか (answer_flag) のフラグを判定する.1 であれば, 後述の output 関数で結果を出力する.1 でなければ, 二進数の 4 つの入力を十進数に変換して sum に格納する.sum の値によって条件分岐する.sum が 15 であれば 3.3 で述べる combine 関数によって入力処理を行う. このとき, 入力処理が正常に終了すれば計算式が記憶されている配列の 1 番目の要素には演算子に相当する数値が格納され,4 番目以降はすべて-1 になるという仕様にしたため, 条件分岐でそれらを判定し,3.4 で述べる計算処理を実行する.sum の値が 15 以外ならば memory に入力した数式を格納していく. 数値を入力していくたびに input_counter を 1 増やし,7 セグメント LED を入力した数値に光らせる. 今回の仕様上三桁と三桁の四則演算が最大の入力数になるので入力した数値が 8 個になると強制的にエラーのフラグを立てる.

3.2 エラー処理

1 で述べた仕様のように, エラーが起きる可能性のある所に条件分岐を作り,millis 関数で現在の時間を取得してグローバル変数 now に格納し,error_flag を 1 にする. そうすることで,loop 関数の最初の条件分岐で error 関数を呼び出しすことができる. これは非トラッキング処理をエラー関数ではエラー関数内で再び millis 関数を呼び出してエラーが発生した段階の時間を格納している now との差が 1000 以下の時, つまりエラーが発生してから 1 秒以下の時は 7 セグメント LED に 14 を表示させる.1 秒経つと全消灯にして使用しているグローバル変数の初期化を行う.

3.3 入力処理

0 から始まるカウンタ i を input_counter の値と等しくなるまで while 文を実行する.while 文の内容は, まず memory[i] の値が 10 未満かどうかを判定し,i 番目に格納されているものが数値であるのか演算子であるのかを判定する. そして, 数値であれば memory[i+1] が 10 未満かどうかを判定する. これにより,21 のような二桁の数値が入力されたときの条件分岐を作成した. 同様にして memory[i+2] を判定して三桁の数値の時の条件分岐を作成した.2 桁であったときには memory[i] に memory[i] を 10 倍したものと memory[i+1] を足し合わせたものを格納し memory 全体を一つずつ左に寄せる. 同様に,3 桁であったときには memory[i] に memory[i] を百倍したものと memory[i+1] を 10 倍したものと memory[i+2] を足し合わせたものを格納し memory 全体を二つ左に寄せる. これらは input_counter の値を超えない範囲で行われる. また, 左にずらした分だけ input_counter の値も減らされる. そして while 文が終了した後に memory の input_counter の値のインデックスから最後まで-1 が格納される. これにより loop 関数での入力処理が正しく行われたかどうかを条件分岐で正常に判定することができる.-1 を選んだ理由は, どのように入力をしても負の値を入力することはできないからである.

3.4 計算

3.3 の入力処理により,memory には順に数値, 演算子, 数値が格納されているはずなので,memory[1] の値によってどの演算を実行するかを判定して計算結果を long 型変数 answer に格納した. 今回の仕様では数

値は三桁までなので answer がとりうる最大の値は 999*999=998001 だが,int 型を使うと 32767 以上の値はオーバーフローをしてしまうので long 型を使用した. 引き算は仕様により絶対値で表すので abs 関数を利用した. 掛け算では memory[0] と memory[1] を long 型に型変換して計算した. そして answer の値を temp という変数にコピーし、10 で何回割ることができるかを確認する. これにより, 計算結果が何桁であるのかを調べることができる. そして 10 で割った余りを answer_memory に格納する.

3.5 結果出力

結果を出力するのはスイッチを開放した瞬間であるため,answer フラグを立てて output 関数を loop することができるようにした.output 関数では最初に表示した回数を数える count が桁数未満かどうかを判定する. 条件式が真であれば answer_memory[count] の数値を二進数に変換し 7 セグメント LED に表示させて count の値を 1 増やす. 偽であれば answer_flag を 0 にして 7 セグメント LED を全消灯にして memory や answer_memory を初期化する.

4 考察と工夫点

今回私が電卓を実装するにあたって工夫した点はエラーが発生した際に自動で初期状態に復帰するようにした点である.7 セグメント LED を 14 の値のままで光らせ続けると, エラー処理によって光っているのか何らかのプログラムのミスによって 14 として光っているのかを判別できる. また, エラーが発生するのは数式を間違って入力したときであるため, 直後に正しい数式を入力する可能性が高いので初期状態に戻ることで素早く次の操作に移ることができる.

5 感想

今回の電卓の課題を実装して感じたことは, ループ関数があるときにはフラグを使うと設計がしやすいということである. フラグを使うことによって自然に非トラッキング処理を実装することができた.

6 謝辞

本課題を実装するにあたって講義時間中の質問応答や動作確認などのご協力をしていただき有難うございました. 今後の課題 3,4 とも宜しくお願いいたします.

参考文献

[1] 情報科学実験 B 指導書

[2] https://www.marutsu.co.jp/pc/static/large_order/1405_311_ph 2024/05/16 アクセス

[3] https://akizukidenshi.com/catalog/g/g110923/ 2024/05/16 アクセス