1　目的

H8 マイコンの割り込みについて理解を深め、割り込み制御を C 言語で記述する方法を学ぶ。本実験では、発展的な課題として「UFO ゲーム」作成に取り組み、キー入力処理と効果音処理を割り込みを用いて実現する。

2　原理

2.1　UFO ゲームの概要

本実験の課題である「UFO ゲーム」は、1980 年代にゲーム電卓としてカシオ計算機から発売された MG-880 の「デジタルインベーダー」が元となっている。

「UFO ゲーム」では、図 1 のようにキャラクタを使ってゲームフィールドを表現する。フィールドは、「:」によって分けられた照準フィールドと戦闘フィールドの 2 つから構成される。

照準フィールドには、敵を打ち落とすための数値が表示され、照準キーを押すたびに「... → 0 → 1 → 2 →... → 8 → 9 → n → 0 →...」のように順に変化する。戦闘フィールドには、フィールドの右側から左に向って敵が押し寄せてくる。押し寄せてくる速さは、ゲームが進行するにつれて徐々に速くなる。敵はインベーダを表す数字の 0~ 9 と、UFO を示す n で表現されている。なお、インベーダは進行するたびに戦闘フィールドの右端に乱数で発生し、UFO は特定の条件下で発生する。

敵の撃墜は、照準フィールドの値を敵を示す値と一致させて発射キーを押すことで行われるが、戦闘フィールドに複数の同じ値をもつ敵が存在する場合には最も照準フィールド寄りの敵 1 つのみが撃墜される。

撃墜された敵は消滅し、撃墜された敵よりも照準側にある敵は右側へ退く。敵の左端が「:」まで到達したときに占領されて、ゲーム終了となる。ゲームの完成見本として、UFO_game.mot が後述の実験用のディレクトリにあるので、各自で検証すること。

2.2　キー状態の判別

今回の実験課題では、キーを押すごとに照準の値を順に変化させなければならないため、キー状態の判別は重要な処理となる。一般に、機械的な接点をもつスイッチの特性として、チャタリング現象が生じる。チャタリング現象は、接点が切り替え時に振動することによって生じ、短い時間 (数 ms から数 10ms 程度) に ON と OFF を繰り返す現象である。その値をそのままマイコンで取り込むと、図 2 のように、一度だけ押したはずが複数回押したように検出されるため、チャタリングの除去が必要となる。

チャタリングの除去にはハードウェア処理による手法とソフトウェア処理による方法があるが、実験ボードのようにキーがマトリクスで実装されている場合はハードウェア処理ではコストがかかるため、一般的にはソフトウェア処理で行われることが多い。

チャタリング処理のアルゴリズムの原理は単純で、入力信号履歴を参照して過去一定期間内の値が全て同じ時に、その値を参照時の値とするものである。一定期間内の値が全て一致しない場合はキー状態が遷移中であると考え、アプリ側で適切に処理する必要がある。

2.3 リングバッファ

キー状態の判別には一定期間内の信号履歴が必要となるため、キーをセンシングするたびに値を保存しておく必要がある。

リングバッファはこのような用途によく用いられ、図 3 のように、通常の配列のような一次元状のバッファの最初と最後をつなぎ合わせることでリング状のバッファを形成する。すなわち、リングバッファの大きさが過去の履歴を保存しておく大きさとなる.

リングバッファのどの位置に最新のデータが入っているかを示すためにポインタを用意し、データの更新を行うたびにポインタも更新する。このとき、リングバッファの実現に配列を利用する場合、ポインタの更新にあたっては配列の最後の次は配列の最初になることに留意しなければならない。このことは、過去の履歴の参照の際にも必要となるので注意すること。

2.4 効果音の生成

効果音を発生させる方法はいくつかあるが、ここでは単純な一定音程で一定の期間の音を生成する方法について考える。一定音程の単音を発生させる最も簡単な方法は、図 4 のように、矩形波を生成させる方法である。矩形波の生成は、一定期間ごとに出力値を変化させることで実現できる。

このとき、単音のみの生成に専念するのであれば一定期間をムダ時間ループで実現すればよいが、本課題では音を生成することによってゲームの進行を妨げることができない。このため、割り込みを音程の半周期ごとにかけることで出力値の変化のタイミングをはかることで、ムダ時間の弊害を取り除くように工夫する。

また、効果音の長さは音長を割り込みの間隔で割ることで、指定時間となる割り込みの回数を知ることで制御を行う。

なお、効果音生成用の割り込み間隔は音程によって変化するため、他のキー処理等の常に一定間隔の割り込みと共用することはできない。このため、効果音生成用に専用のタイマを準備して割り込み制御を行うことが必要である。

3　使用機器

使用機器を以下に示す。

1. H8 マイコン
2. USB ケーブル
3. パーソナルコンピュータ

4　実験方法

本課題では、内蔵されているタイマ 2 つを割り込み発生源として使用する。タイマ 0 を効果音生成専用のタイマとして割り込み優先度をプライオリティ1(高優先度) として用いる。プライオリティ1 の優先度を使用するためには、UI ビットの設定が必要であり。USE_UI(); をプログラムの最初に実行する必要がある。また、タイマ 0 をプライオリティ1 に設定するためには、更に timer_pri_set(ch, pri) で設定を行う必要がある (ch:タイマのチャネル番号、pri:プライオリティ値)。割り込みハンドラは int_imia0() であり、sound.c 内に記述する。

タイマ 1 は常に一定間隔 (1ms) で割り込みを発生し、キーのセンシング・敵の進行速度・時間待ち等の処理を行うものとする。割り込みハンドラは int_imia1() であり、ufo.c に記述する。

課題にあたっては、/home/class/j3/jikken/kouki/no4 以下に課題に必要なファイルを置いてあるので、必ず全てのファイルを各自でコピーして利用すること。

4.1　課題 1 キー読み込みの実装

コピーしたファイル (特に、ufo.c, key.c) をよく読み、key.c に記述されている各関数を完成させなさい。

key_sense() はタイマ 1 の割り込みごとに呼び出され、リングバッファにキーマトリクスの列ごとのスキャンデータを格納するように作成すること。リングバッファは key.c で大域変数として宣言されている keybuf[p][r] であり、p は参照ポインタ、r は列を表すものとする。また、割り込み処理をまたぐリングバッファのポインタは keybufdp を用いること。なお、キースキャンの方法等については、最後のページにある補足のキーマトリクスとセンシングを参照のこと。

key_check(keynum) はチャタリング除去を行った結果を返す。その戻り値は key.c で定義されている KEYOFF(指定キーは押されていない)、KEYON(指定キーが押されている)、KEYTRANS(指定キーは変化中)、KEYNONE(指定キー

は存在しない) のいずれかを正しく返すものとする。また、チャタリング除去で参照するバッファ上の範囲は、key.c で定義されている KEYCHKCOUNT の長さとする。課題ができたら、必ずその時点でコンパイル・実行して動作を確認し、プログラムと動作のチェックを受けること。チェックを受けずに先の課題を行うことは認めない。

4.2 課題 2 効果音生成の実装

コピーしたフアイル (特に、ufo.c, sound.c) をよく読み、sound.c に記述されている各関数を完成させなさい。

sound_beep(hz、msec, vol) は引数 (hz:音程周波数で単位は [Hz], sec:音長で単位は [ms], vol:振幅で矩形波の D/A 上限値) から計算して、音程のための割り込み間隔・音長のための割り込み回数・短形波の振幅を求めて必要な記述を加えること。

sound.c で宣言されている大域変数 timer0_count は割り込み回数を格納し、play_count は指定音長に対する割り込み回数を格納 da_amp は指定振幅を格納するように、それぞれ準備されている。-課題ができたら、必ずその時点でコンパイル・実行して動作を確認し、プログラムと動作のチェックを受けること、チェックを受けずに先の課題を行うことは認めない。

4.3 課題 3 UFO ゲームの発展

以下の課題に取り組み、UFO ゲームを更に発展させる。細部の仕様は適宜定めてよいが、レポートに記述すること。

・課題 3-1

自機数を導入し、占領された場合と、発射キーを押したときに照準値と同じ値の敵がいない場合 (攻撃に失敗した場合) に自機数が減るようにして、自機数が 0 になったときにゲームオーバとなるように変更する。

・課題 3-2

出現する敵数と発射できる弾数を制限し、敵を全部撃破したら面クリアとする。面が進むにつれて条件を厳しくし、占領されたときにゲームオーバとなるように変更する。

5　実験結果

5.1　課題 1 キー読み込みの実装

key.h に宣言されている関数を実装することで、チャタリングを除去して、キーでゲームを操作することが可能になった。

ソースファイルをリスト 1 に、Makefile をリスト 2 にそれぞれ示す。

5.2　課題 2 効果音生成の実装

sound.h に宣言されている関数を実装することで、ボタン操作に応じて効果音がなるようになった。

ソースファイルをリスト 3 に示す。Makefile は課題 1 のリスト 2 と同じである。

5.3　課題 3 UFO ゲームの発展

・課題 3-1

課題の条件を満たすプログラムを作成し、実際に動作することを確認した。

ソースファイルをリスト 4 に示す。Makefile は課題 1 のリスト 2 と同じである。

・課題 3-2

ステージが進むに連れて、敵の数は多くなり、余分な弾数が少なくなるようにした。敵または弾数が 99 以上の場合、表示は 99 のままで 99 未満になるとその数値が表示される仕様にした。表示中の「E:」は敵の残り、「B:」は残り弾数、そして「S:」は現在のステージを表す。

ソースファイルをリスト 5 に示す。Makefile は課題 1 のリスト 2 と同じである。

6　検討課題

・検討課題 1

ufo.c をよく読んで、どのようにして「今、キーが押された」かを検出しているかについて、説明しなさい。

1 ループ前のキーの状態を保持し、現在のキーの状態と比較することにより実現されている。1 ループ前に押されていない、かつ現在押されていればキーは今押されたことになる。

・検討課題 2

補足の図 5 キーマトリクス回路において、キースイッチと直列にダイオードが挿入されていないとどのような不具合が生ずるか、説明しなさい。

読み取ろうとしている行のいずれかのキーを押した状態で、そのキーと同じ列の他のキーを 1 つ以上押すと、5V 端子と GND が短絡された状態になり、大きな電流が流れるため危険になる。

・検討課題 3

sound.c で実装したような割り込み処理による効果音生成において、効果音の音程が高くなると、どのような悪影響が生じてくるかについて検討せよ。

効果音のための割り込みの回数が多くなり、メインの処理、キーの処理に充てられる時間が減り、キーの取りこぼしが発生したり反応が遅くなったりして快適に遊べなくなる恐れが出てくる。

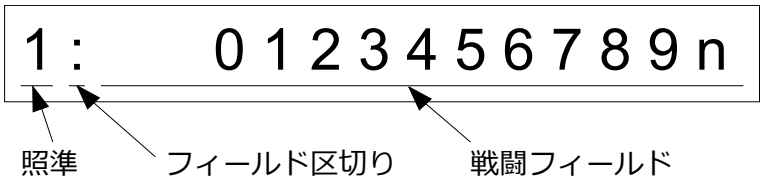

図 1: UFO ゲームのフィールド

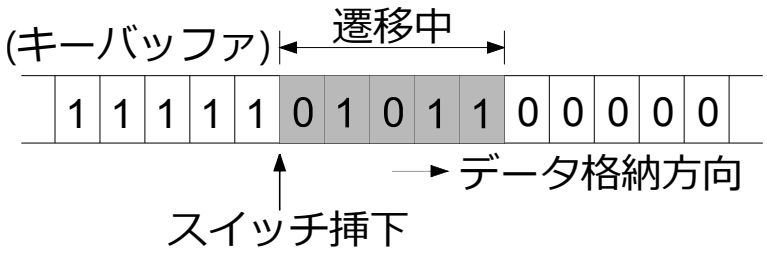

図 2: チャタリング発生時のデータバッファの様子

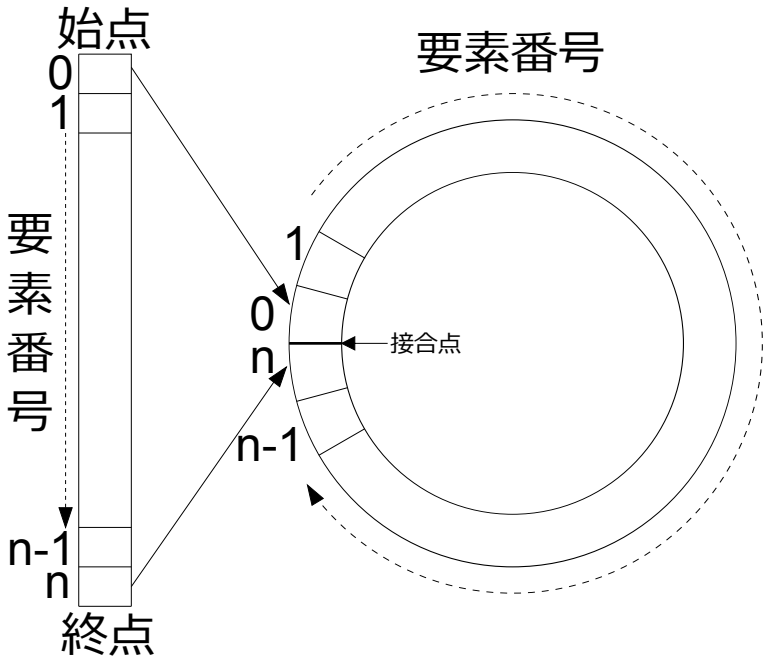

図 3: リングバッファの構成

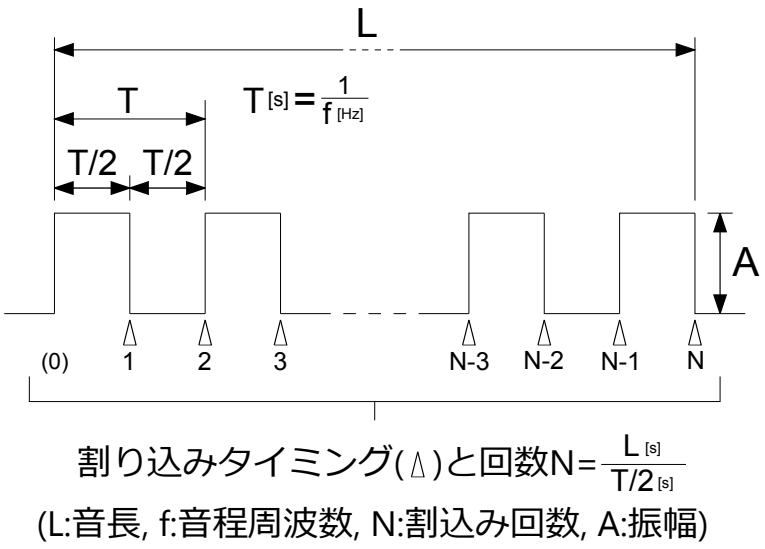

図 4: 効果音の波形

リスト 1: key.c

```
#include "h8-3052-iodef.h"

#define KEYBUFSIZE 10  /* キーバッファの大きさ */
#define KEYCHKCOUNT 5  /* キーの連続状態を調べるバッファ上の長さ   */
/* ↑キーバッファの大きさよりも小さくすること */
/* 余裕が少ないと正しく読めないことがある    */
#define KEYROWNUM 4    /* キー配列の列数縦に並んでいる個数() */
#define KEYCOLNUM 3    /* キー配列の行数横に並んでいる個数() */
#define KEYMINNUM  1   /* キー番号の最小値 */
#define KEYMAXNUM 12   /* キー番号の最大値 */
#define KEYNONE   -1   /* 指定したキーがない */
#define KEYOFF     0   /* 指定したキーはずっと離されている状態 */
#define KEYON      1   /* 指定したキーはずっと押されている状態 */
#define KEYTRANS   2   /* 指定したキーは遷移状態 */

// キースキャンを行って、状態を調べる関数群
// 一定時間（数程度）毎にms keysense() を呼び出すことが前提
// 任意のキー状態を読み出すには key_check() を呼び出す

/* タイマ割り込み処理のため, バッファ関連は大域変数として確保 */
/* これらの変数は key.c 内のみで使用されている                 */
int keybufdp; /* キーバッファ参照ポインタ */
unsigned char keybuf[KEYBUFSIZE][KEYROWNUM]; /* キーバッファ */

void key_init(void);
void key_sense(void);
int key_check(int keynum);

void key_init(void)
  /* キーを読み出すために必要な初期化を行う関数 */
  /* PA4-6 が LCD と関連するが, 対策済み       */
{
  int i,j;

  PADR = 0x0f;        /* PA0-3 はアクティブ0, PA4-6 はアクティブ1 */
  PADDR = 0x7f;       /* PA0-3 はキーボードマトリクスの出力用 */
  /* PA4-6 は制御LCD(E,R/W,RSの出力用) */
  P6DDR = 0;          /* P60-2 はキーボードマトリクスの入力用 */
  /* P63-6 はのバス制御として固定CPUモードの時(6) */
  keybufdp = 0;
```

```
41    /* キーバッファのクリア */
42    for (i = 0; i < KEYBUFSIZE; i++){
43      for (j = 0; j < KEYROWNUM; j++){
44        /* ここで何もキーが押されていない状態にバッファ(keybufを初期化) */
45        /* キーが押されていないときにビットがとなることに注意すること1 */
46        keybuf[i][j] = 0x07;
47      }
48    }
49  }
50
51  void key_sense(void)
52    /* キースキャンしてキーバッファに入れる関数            */
53    /* 数  ms 程度に一度, タイマ割り込み等で呼び出すこと */
54    /* 大域変数    keybuf はキーデータを格納するバッファ  */
55  {
56    /* リングバッファポインタ制御 */
57    /* ここにバッファポインタ(keybufdpの更新を書く) */
58    /*   ・バッファポインタが最新のスキャンデータを指すようにすること */
59    /*   ・リングバッファのつなぎ目の処理を忘れないこと */
60    keybufdp = (keybufdp-1+KEYBUFSIZE) % KEYBUFSIZE;
61
62    /* キースキャン */
63    /* ここでキー列ごとにキースキャンしたデータをそのままキーバッファに格納する */
64    /* キー列番号は、~ の列、~ の列、~ の列、~ の列0:131:462:793:*# とする */
65    /* 各キー列のキーデータは keybufバッファポインタキー列番号[][] に格納する */
66    /*   ・~ だけを書き換えるように注意することPA0PA3他のビットの変化禁止() */
67    /*   ・~ だけを読むように注意することP60P62他のビットはにする(0) */
68
69    /*今回使われているのはkeyだけ*0#
70    //key 1,2,3
71    PADR |= 0x07;
72    PADR &= 0xf7;
73    keybuf[keybufdp][0] = P6DR & 0x07;
74
75    //key 4,5,6
76    PADR |= 0x0b;
77    PADR &= 0xfb;
78    keybuf[keybufdp][1] = P6DR & 0x07;
79
80    //key 7,8,9
81    PADR |= 0x0d;
82    PADR &= 0xfd;
83    keybuf[keybufdp][2] = P6DR & 0x07;
84    */
85
86    //key *,0,#
87    PADR |= 0x0e;
88    PADR &= 0xfe;
89    keybuf[keybufdp][3] = P6DR & 0x07;
90  }
91
92  int key_check(int keynum)
93    /* キー番号を引数で与えると, キーの状態を調べて返す関数                    */
94    /* キー番号(keynumは) 1-12 で指定回路図の( sw1-sw12 に対応)                */
95    /* 基板上の 1-9 のキーは sw1-sw9 に対応している                            */
96    /* 基板上の *,0,# のキーは sw10,sw11,sw12 にそれぞれ対応している          */
97    /* 戻り値は, KEYOFF, KEYON, KEYTRANS, KEYNONE のいずれか                  */
98    /* チェック中の割り込みによるバッファ書き換え対策はバッファの大きさで対応 */
99  {
100   int r;
101   int row, pos;
102
103   /* 最初にキー番号の範囲をチェックする */
104   if ((keynum < 1) || (keynum > KEYMAXNUM))
105     r = KEYNONE; /* キー番号指定が正しくないときはを返すKEYNONE */
106   else {
107     /* ここでキー番号からキー列番号とデータのビット位置を求める */
108     /* キー列番号がわかると配列の参照ができる */
109     /* データのビット位置がわかれば、指定されたキーのON/がわかるOFF */
110     row = (keynum-1) / (KEYROWNUM-1);
111     pos = (keynum-1) % (KEYROWNUM-1);
112
113     /* ここで宣言された長さ(KEYCHKCOUNT分だけキーの状態を調べる) */
114     /*   ・リングバッファのつなぎ目の処理を忘れないこと */
```

```
    /*   ・途中でキースキャン割り込みが生じても矛盾しない処理を行うこと */
    /* 指定キーが全てなら、全てなら、それ以外はONKEYONOFFKEYOFFKEYTRANS とする*/
    int temp = keybufdp;

    if(keybuf[temp][row] & (1 << pos)){
      r = KEYOFF;
    }else{
      r = KEYON;
    }

    int i;
    for(i = 0; i < KEYCHKCOUNT-1; i++){
      if((keybuf[(temp+i)%KEYBUFSIZE][row] & (1 << pos)) ^
          (keybuf[(temp+i+1)%KEYBUFSIZE][row] & (1 << pos))){
        r = KEYTRANS;
        break;
      }
    }
  }
  return r;
}
```

リスト 2: Makefile

```
# H8/3052 の雛型  Makefile (2008.5.29 和崎)
#   1. 必要な設定を変更して、違うファイル名で保存する例： (make-test)
# TARGET = , SOURCE_C = , SOURCE_ASM = を指定する
# リモートデバッキングのときは、GDBREMOTE_DBG = true とする
# その他は通常、変更の必要はない
#   2. make -f 名makefile で make する例： (make -f make-test)

# 生成するファイルとソースファイルの指定
# 1. 生成するオブジェクトのファイル名を指定
TARGET = ufo.mot
# 2. 生成に必要なのファイル名を空白で区切って並べるC
SOURCE_C = ufo.c lcd.c timer.c key.c sound.c da.c random.c
# 3. 生成に必要なアセンブラのファイル名を空白で区切って並べる
# スタートアップルーチンは除く()
SOURCE_ASM =

# 生成するオブジェクトの種類を指定
# ※の項目は通常変更する必要がない()
#
# 1. によるリモートデバッキング指定GDB
# true : 指定する    その他：指定しない
REMOTE_DBG =

# 2. 上デバッグまたは化指定RAMROM ※
# ram : 上で実行RAM rom : 化ROM
ON_RAM = ram

# 3. 使用領域の指定RAM ※
# ：化→プログラムとスタックは外部を使用extRAMRAM
# 化→スタックは外部      ROMRAM
# ：化→プログラムとスタックは内部を使用intRAMRAM
# 化→スタックは内部      ROMRAM
# 指定なし：化→プログラムは外部、スタック変更なしRAMRAM
#   化→スタックは外部  ROMRAM
RAM_CAP = ext

# 4. によるデバッグを行うかどうかの指定GDB ※
USE_GDB = true

# 計算機環境依存項目の指定
# 使用する環境にあわせて変更、通常は変更の必要なし()
#
# 0. 計算機システムの指定
# ：情報工学科計算機システムjcomp
# 指定なし：以下のパスの設定に従う
COMP_SYS = jcomp
# COMP_SYS =
#
```

```
# 1. クロス環境のバイナリが置かれているパスの指定
# 回路システム実験室ではこちらのディレクトリ
#   CMD_PATH = /usr/local/H8/bin
# デフォルト指定
CMD_PATH = /usr/local/h8/bin
#
# 2. リンカスクリプト、スタートアップルーチン、その他ライブラリ
# デフォルト指定
LIB_PATH = /home/wasaki/h8/standard-set/3052
#
# 情報工学科計算機システムの指定があるとき、パスは以下の設定になる
ifeq ($(COMP_SYS), jcomp)
  CMD_PATH = /usr/local/bin
  LIB_PATH = /home/class/common/H8/lib
endif
#
# 3. クロスコンパイラ関係
# デフォルト指定
CC = $(CMD_PATH)/h8300-hms-gcc
LD = $(CMD_PATH)/h8300-hms-ld
OBJCOPY = $(CMD_PATH)/h8300-hms-objcopy
SIZE = $(CMD_PATH)/h8300-hms-size

#
# ターゲット指定
#
TARGET_COFF = $(TARGET:.mot=.coff)
MAP_FILE = $(TARGET:.mot=.map)

#
# 出力フォーマット
# binary : binary, srec : Motorola S record, ihex : Intel Hex
#
OUTPUT_FORMAT = -O srec --srec-forceS3

#
# コンパイラオプション
#
# インクルードディレクトリの追加("*****.h"指定のみ有効)
INCLUDES = -I./
# コンパイラオプションの指定
# -：mhH8/300シリーズ指定H
# -：条件分岐コードの最適化mrelax
# -：型変数のビット数指定mint32int
# -：の最適化レベルの指定O2gcc
# -：コンパイル時の警告メッセージの選択Wall全て()
CFLAGS = -mh -mrelax -mint32 -O2 $(INCLUDES) -Wall

#
# 指定に合わせたスタートアップルーチンとリンカスクリプトの選択
#

ifeq ($(REMOTE_DBG), true)
  USE_GDB = true
  ON_RAM = ram
  RAM_CAP =
endif

ifeq ($(USE_GDB), true)
  CFLAGS := $(CFLAGS) -g
endif

ifeq ($(ON_RAM), ram)
  LDSCRIPT = $(LIB_PATH)/h8-3052-ram.x
  STARTUP = $(LIB_PATH)/ramcrt.s
  ifeq ($(RAM_CAP), int)
        LDSCRIPT = $(LIB_PATH)/h8-3052-ram8k.x
    STARTUP = $(LIB_PATH)/ramcrt-8k.s
  endif
  ifeq ($(RAM_CAP), ext)
    LDSCRIPT = $(LIB_PATH)/h8-3052-ram.x
    STARTUP = $(LIB_PATH)/ramcrt-ext.s
  endif
  ifeq ($(REMOTE_DBG), true)
```

```
    LDSCRIPT = $(LIB_PATH)/h8-3052-ram-dbg.x
    STARTUP = $(LIB_PATH)/ramcrt-dbg.s
  endif
else
  ifeq ($(RAM_CAP), int)
    LDSCRIPT = $(LIB_PATH)/h8-3052-rom8k.x
    STARTUP = $(LIB_PATH)/romcrt-8k.s
  else
        LDSCRIPT = $(LIB_PATH)/h8-3052-rom.x
    STARTUP = $(LIB_PATH)/romcrt-ext.s
  endif
endif

#
# リンク時のコンパイラオプションの指定
# -T ：リンカスクリプトファイルの指定filename
# -：標準のスタートアップを使用しないnostartfiles
# -Wlパラメータ…：リンカに渡すパラメータ指定,,
#   -Map ：メモリマップをに出力mapfilenamemapfilename
LDFLAGS = -T $(LDSCRIPT) -nostartfiles -Wl,-Map,$(MAP_FILE)

#
# オブジェクトの指定
#
OBJ = $(STARTUP:.s=.o) $(SOURCE_C:.c=.o) $(SOURCE_ASM:.s=.o)

#
# サフィックスルール適用の拡張子指定
#
.SUFFIXES: .c .s .o

#
# ルール
#
$(TARGET) : $(TARGET_COFF)
  $(OBJCOPY) -v $(OUTPUT_FORMAT) $(TARGET_COFF) $(TARGET)

$(TARGET_COFF) : $(OBJ)
  $(CC) $(CFLAGS) $(LDFLAGS) $(OBJ) -o $(TARGET_COFF)
  $(SIZE) -Ax $(TARGET_COFF)

clean :
  rm -f *.o $(TARGET) $(TARGET_COFF) $(MAP_FILE)

#
# サフィックスルール
#
.c.o:
  $(CC) -c $(CFLAGS) $<
.s.o:
  $(CC) -c $(CFLAGS) $<
```

リスト 3: sound.c

```
#include "h8-3052-iodef.h"
#include "timer.h"
#include "da.h"
#include "h8-3052-int.h"

// 単音を一定時間鳴らすための関数群
// タイマを使って音程の周期で割り込みを起こす01/2
// 音の長さは割り込み回数をカウントして測る

void int_imia0(void);

/* sound.の中だけで閉じている大域変数、割り込みハンドラ用c */
unsigned int timer0_count, play_count;
unsigned char da_amp;

void sound_init(void)
     /* 音を鳴らすための初期化                */
     /* D/変換用の初期化とスピーカの切り替えA */
```

```
{
  da_init();                /* の初期化DA */
  speaker_switch(SPEAKER); /* スピーカとして使用 */
}

void sound_beep(int hz,int msec,int vol)
     /* タイマの割り込みを使って単音を一定の長さだけ鳴らすための関数0   */
     /* 引数は、音程:hz, 音長:msec, 音量:vol                              */
     /* timer0_count, play_cont, da_amp は割り込みハンドラで使用     */
{
  unsigned int int_time;

  timer0_count = 0;       /* 割り込み回数用カウンタの初期化 */

  /* ここで割り込み周期単位はμ([sを求めて]) int_time に入れる */
  /* 割り込み周期は音程周期の半分 */
  int_time = 1000000/(2*hz);


  /* ここで指定音長となる割り込み回数を求めて play_count に入れる */
  /* 単位に注意して音長が割り込み何回分かを求める */
  play_count = msec*1000 / int_time;

  /* ここで指定音量になるように da_amp にセットする */
  /* 割り込みハンドラに渡すために大域変数に入れる */
  da_amp = vol;

  timer_set(0,int_time);  /* 音程用割り込み周期のセット */
  timer_start(0);         /* タイマスタート0 */
}

#pragma interrupt
void int_imia0(void)
     /* 音程を出すための割り込みハンドラ */
     /* 使用する大域変数と役割は以下の通り */
     /*   timer0_count は割り込み回数を数えるカウンタ */
     /*   play_cont は指定音長になるときの割り込み回数 */
     /*   da_amp はD/の出力上限値A  */
{
  /* ここで、割り込み回数をインクリメントする */
  timer0_count++;

  /* ここで、割り込みがかかる度にD/の出力を上限値か下限値に切替えるA */
  da_out(0, da_amp*(timer0_count%2));

  /* ここで、タイマカウントが音長カウントに達したらタイマストップする */
  /* タイマストップしたら割り込みはかからなくなる */
  if(timer0_count > play_count)
    timer_stop(0);


  /* 再びタイマ割り込みを使用するために必要な操作       */
  timer_intflag_reset(0);  /* タイマの割り込みフラグをクリア0 */
  ENINT();                 /* を割り込み許可状態にCPU */
}
```

リスト 4: 課題 3-1 の ufo.c

```
#include "h8-3052-iodef.h"
#include "h8-3052-int.h"

#include "lcd.h"
#include "random.h"
#include "timer.h"
#include "key.h"
#include "da.h"
#include "sound.h"

/* フラグ定数 */
#define TRUE  1
#define FALSE 0

```

```
/* 表示に関する定数 */
  /* の横方向桁数LCD */
#define MAXDIGITNUM 16
  /* ゲームに使う表示桁数 */
#define MAXINVNUM   16

/* 敵の進行スピードに関する定数（単位は）ms */
  /* 進行時間間隔 */
#define INITSPEED   2000
  /* スピード上昇定数 */
#define SPEEDUP     10

/* 効果音に関する定数 */
  /* TONE_XXX は音程（単位は）Hz */
  /* LONG_XXX は音長（単位は）ms */
#define TONE_TARGET 83
#define LONG_TARGET 100
#define TONE_MISS   50
#define LONG_MISS   300
#define TONE_HIT    250
#define LONG_HIT    200
#define TONE_DEF    63
#define LONG_DEF    200

/* 得点表示の長さの定数（単位は）ms */
#define WAITFEWSEC  5000

/* 割り込みハンドラで処理する変数 */
volatile unsigned long n_time, speed_count, speed;
volatile int shift_flag;

int main(void);
char *ntos(unsigned int n, char *s);
int game_start(void);
void effect(char c);
void int_imia0(void);
void int_imia1(void);

int main(void)
     /* ゲームUFO電卓ゲーム風() */
{
  unsigned int score, high_score;
  char score_s[MAXDIGITNUM+1];

  /* 初期化 */
  ROMEMU();           /* エミュレーションをROMON */
  USE_UI();           /* 多重割り込み制御のためにビットを使用UI */
  timer_pri_set(0,1); /* タイマの割り込みをプライオリティ優先に設定01() */
  random_init();      /* 乱数発生の初期化 */
  lcd_init();         /* 表示器の初期化LCD */
  key_init();         /* キースキャンの初期化 */
  timer_init();       /* タイマの初期化 */
  sound_init();       /* 効果音の初期化 */
  timer_set(1,1000);  /* タイマは音長キーセンス速度調整時間待ち用1/// */
  timer_start(1);     /* タイマスタート1 */
  ENINT();            /* 全割り込み受付可 */
  /* LCD にメニュー表示 */
  lcd_cursor(0,0);
  lcd_printstr(" 0:Game Start   ");
  lcd_cursor(0,1);
  lcd_printstr(" *:High Score   ");
  high_score = 0;
  while(1){
    if (key_check(10) == KEYON){     /* キーハイスコア表示*: */
      lcd_cursor(0,0);                    /* LCD にハイスコア表示 */
      lcd_printstr(" High Score is  ");
      lcd_cursor(0,1);
      lcd_printstr("                ");
      lcd_cursor(0,1);
      lcd_printstr(ntos(high_score,score_s));
    }
    if (key_check(10) == KEYOFF){     /* キーを離したらメニュー表示* */
      lcd_cursor(0,0);                      /* LCD にメッセージ表示 */
      lcd_printstr(" 0:Game Start   ");
```

```
      lcd_cursor(0,1);
      lcd_printstr(" *:High Score   ");
    }
    if (key_check(11) == KEYON){      /* キーゲームスタート0: */
      lcd_cursor(0,0);                /* LCD に操作方法表示 */
      lcd_printstr("*:Sight  0:Trig.");
      score = game_start();           /* ゲームスタート */
      lcd_cursor(0,1);                /* 得点表示欄のクリア */
      lcd_printstr("                ");
      if (score > high_score){        /* ハイスコアのとき */
  high_score = score;                /* ハイスコア登録 */
  lcd_cursor(0,0);                   /* ハイスコア表示 */
  lcd_printstr(" High Score !!! ");
  lcd_cursor(0,1);
  lcd_printstr(ntos(high_score,score_s));
      } else {                        /* ハイスコアでないとき*/
  lcd_cursor(0,0);                   /* スコアを表示 */
  lcd_printstr(" Your Score ... ");
  lcd_cursor(0,1);
  lcd_printstr(ntos(score,score_s));
      }
      n_time = 0;
      while (n_time < WAITFEWSEC);    /* 得点表示後にちょっと待つ */
      lcd_cursor(0,0);                  /* LCD にメッセージ表示 */
      lcd_printstr(" 0:Game Start   ");
      lcd_cursor(0,1);
      lcd_printstr(" *:High Score   ");
    }
  }
}

char *ntos(unsigned int n, char *s)
     /* 数値を文字列に変換する関数 */
     /* 引数は、整数値：(>0)n, 変換文字列が入る文字列変数ポインタ：s */
     /* 戻り値は、文字列変数ポインタ */
{
  int count, i;
  char st[MAXDIGITNUM+1];
  char d;

  count = 0;
  do {
    d = n % 10;
    n = (n - d) / 10;
    st[count] = d + '0';
    count++;
  } while (n > 0);
  for (i = 0; i < count; i++)
    s[i] = st[count - 1 - i];
  s[count] = '\0';
  return s;
}

int game_start(void)
     /* ゲームの本体UFO                                          */
     /* 戻り値: ゲームスコア                                     */
     /* 処理の高速化のためになるべく関数化しないで書いてある     */
     /* 関数を呼び出すと多くのレジスタを退避するし,               */
     /* 外部メモリのアクセスはビット幅でかつ遅い8                 */
     /* 但し, きれいに関数化しても差障りはないかもしれない       */
{
  int target, score, gameover, hit, i, j;
  int prev_key10, prev_key11, key10, key11;
  unsigned int ds;
  char t, nc;
  char disp[MAXINVNUM+1]; /* [0: 1234567890n] のような画面イメージ */
                          /* 照準区切りインベーダと ,,UFO */
  char stock_disp[] = "Nokori: ";
  int stock = 3;
  stock_disp[sizeof(stock_disp)-2] = stock + '0';

  target = 0; t = '0'; score = 0;     /* 照準とスコアの初期化 */
  speed_count = 0; speed = INITSPEED; /* プレイ速度の初期化 */
  shift_flag = FALSE;                 /* 敵の前進フラグの初期化 */
```

```
163   disp[0] = '0'; disp[1] = ':';        /* 表示フィールドの初期化 */
164   for (i = 2; i < MAXINVNUM; i++) disp[i] = ' ';
165   disp[MAXINVNUM] = '\0';
166   gameover = FALSE;        /* ゲーム終了フラグの初期化 */
167   key10 = key_check(10);  /* キーバッファ照準キーの初期化() */
168   prev_key10 = KEYOFF;
169   key11 = key_check(11);  /* キーバッファ発射キーの初期化() */
170   prev_key11 = KEYOFF;
171   lcd_cursor(0, 0);
172
173   lcd_cursor(0,1);         /* 初期画面の表示 */
174   lcd_printstr(disp);
175   lcd_cursor(0, 0);
176   lcd_printstr("                ");
177   lcd_cursor(0, 0);
178   lcd_printstr(stock_disp);
179   while (gameover == FALSE){ /* ゲームオーバでなければループ */
180     /* キーの立上りを検出するための前回チェック時のキー状態を記憶 */
181     /* キーバッファ照準キー処理() */
182     if (key10 != KEYTRANS) prev_key10 = key10; /* 遷移中は前々回の値そのまま */
183     key10 = key_check(10);
184     /* キーバッファ発射キー処理() */
185     if (key11 != KEYTRANS) prev_key11 = key11; /* 遷移中は前々回の値そのまま */
186     key11 = key_check(11);
187     if ((prev_key10 == KEYOFF) && (key10 == KEYON)){
188       /* 照準キーが押されたときの処理 */
189       /* 照準は 0->1->...->8->9->n->0->... と順に変化する */
190       target++;                          /* 照準 +1 */
191       if (target > 10) target = 0;       /* の次はUFO0 */
192       if (target < 10) t = '0' + target;  /* 照準がでないときのキャラUFO */
193       else t = 'n';                      /* 照準がのときのキャラUFO */
194       disp[0] = t;                       /* 照準値を表示にセット */
195       effect('t');                       /* 効果音を鳴らす */
196       /* フィールドの表示 */
197       lcd_cursor(0,1);
198       lcd_printstr(disp);
199     }
200     if ((prev_key11 == KEYOFF) && (key11 == KEYON)){
201       /* 発射キーが押されたときの処理 */
202       hit = FALSE;  /* ヒット判定フラグの初期化 */
203       i = 2;        /* 最も左の敵表示位置 */
204       while ((i < MAXINVNUM) && (hit != TRUE)){ /* 列の左から命中を探す */
205   if (disp[i] == t){    /* 照準と一致するキャラか? */
206     hit = TRUE;           /* ヒット判定    */
207     ds = MAXINVNUM - i;   /* 基本得点の計算砲台に近い程大  () */
208     score = score + ds;   /* 得点追加    */
209     if (target > 9) score = score + ds * 2;  /* なら倍の得点UFO3 */
210     for (j = i; j > 2; j--) disp[j] = disp[j-1];  /* 命中左側を右寄せ */
211     disp[2] = ' ';        /* 最も左は常に空白を詰める */
212     /* フィールドの表示 */
213     lcd_cursor(0,1);
214     lcd_printstr(disp);
215   }
216   i++;  /* 探索位置を +1 */
217       } /* ヒット判定があるか右端まで調べたら, 探索終了 */
218       if (hit == TRUE){
219   effect('s'); /* 命中時の効果音 */
220       }else{
221   effect('m');              /* 失敗時の効果音 */
222   stock--; /*ziki wo herasu*/
223   stock_disp[sizeof(stock_disp)-2]--; /*stock_disp[sizeof(stock_disp)-1] = stock + '0 is better form*/
224   if(stock < 0){
225     gameover = TRUE;
226     continue;
227   }
228   lcd_cursor(0,0);
229   lcd_printstr(stock_disp);
230       }
231     }
232     /* 敵が前進するタイミングのときの処理 */
233     if (shift_flag == TRUE){          /* 前進フラグが立っているなら */
234       if (disp[2] != ' ') disp[1] = disp[2]; /* 侵略時のみ区切りに侵入 */
235       for (i = 2; i < MAXINVNUM; i++)   /* 敵全体を左につシフト1 */
236   disp[i] = disp[i + 1];
```

```
237         if (score % 10 == 1) nc = 'n';     /* スコアの最下位がなら1出現UFO */
238         else nc = (random() % 10) + '0';   /* それ以外はランダムな数字 */
239         disp[MAXINVNUM - 1] = nc;          /* 右端に新キャラを入れる */
240         shift_flag = FALSE;                /* 前進フラグの消去 */
241         lcd_cursor(0,1);                   /* フィールドの表示 */
242         lcd_printstr(disp);
243       }
244       if (disp[1] != ':') gameover = TRUE; /* 侵略されたらゲームオーバ */
245     }
246     return score; /* 得点を返す */
247   }
248
249   void effect(char c)
250        /* 効果音を発生するための関数 */
251        /* 引数: c 効果音の種類を指定 */
252   {
253     unsigned int t;
254     unsigned long p;
255
256     switch (c) {        /* 種類によって音程と音長を変えられる */
257     case 't':             /* 照準を動かしたとき */
258       t = TONE_TARGET;
259       p = LONG_TARGET;
260       break;
261     case 'm':             /* ミスしたとき */
262       t = TONE_MISS;
263       p = LONG_MISS;
264       break;
265     case 's':             /* ヒットしたとき */
266       t = TONE_HIT;
267       p = LONG_HIT;
268       break;
269     default:              /* その他のとき*/
270       t = TONE_DEF;
271       p = LONG_DEF;
272     }
273     sound_beep(t,p,32);
274   }
275
276   #pragma interrupt
277   void int_imia1(void)
278        /* キーセンス・速度調整・時間待ち用のタイマ割り込みハンドラ1 */
279   {
280     ENINT1();          /* 音程用の割り込み処理を優先させる */
281     key_sense();       /* キーセンス */
282     n_time++;          /* 時間待ち用のカウンタ +1 */
283     speed_count++;                  /* スピード調整用カウンタ +1 */
284     if (speed_count >= speed){      /* 設定値になったら */
285       speed_count = 0;              /* カウンタ初期化    */
286       speed = speed - SPEEDUP;      /* スピードアップ    */
287       shift_flag = TRUE;            /* 前進フラグ    ON */
288     }
289
290     /* 再びタイマ割り込みを使用するために必要な操作       */
291     /* タイマの割り込みフラグをクリアしないといけない  1 */
292     timer_intflag_reset(1);
293
294     ENINT();           /* を割り込み許可状態にCPU */
295   }
```

リスト 5: 課題 3-2 の ufo.c

```
1   #include "h8-3052-iodef.h"
2   #include "h8-3052-int.h"
3
4   #include "lcd.h"
5   #include "random.h"
6   #include "timer.h"
7   #include "key.h"
8   #include "da.h"
9   #include "sound.h"
10
```

```
/* フラグ定数 */
#define TRUE  1
#define FALSE 0

/* 表示に関する定数 */
  /* の横方向桁数LCD */
#define MAXDIGITNUM 16
  /* ゲームに使う表示桁数 */
#define MAXINVNUM   16

/* 敵の進行スピードに関する定数（単位は）ms */
  /* 進行時間間隔 */
#define INITSPEED   2000
  /* スピード上昇定数 */
#define SPEEDUP     10

/* 効果音に関する定数 */
  /* TONE_XXX は音程（単位は）Hz */
  /* LONG_XXX は音長（単位は）ms */
#define TONE_TARGET 83
#define LONG_TARGET 100
#define TONE_MISS   50
#define LONG_MISS   300
#define TONE_HIT    250
#define LONG_HIT    200
#define TONE_DEF    63
#define LONG_DEF    200

/* 得点表示の長さの定数（単位は）ms */
#define WAITFEWSEC  5000

/* 割り込みハンドラで処理する変数 */
volatile unsigned long n_time, speed_count, speed;
volatile int shift_flag;

int main(void);
char *ntos(unsigned int n, char *s);
int game_start(void);
void effect(char c);
void int_imia0(void);
void int_imia1(void);

int main(void)
     /* ゲームUFO電卓ゲーム風() */
{
  unsigned int score, high_score;
  char score_s[MAXDIGITNUM+1];

  /* 初期化 */
  ROMEMU();           /* エミュレーションをROMON */
  USE_UI();           /* 多重割り込み制御のためにビットを使用UI */
  timer_pri_set(0,1); /* タイマの割り込みをプライオリティ優先に設定01() */
  random_init();      /* 乱数発生の初期化 */
  lcd_init();         /* 表示器の初期化LCD */
  key_init();         /* キースキャンの初期化 */
  timer_init();       /* タイマの初期化 */
  sound_init();       /* 効果音の初期化 */
  timer_set(1,1000);  /* タイマは音長キーセンス速度調整時間待ち用1/// */
  timer_start(1);     /* タイマスタート1 */
  ENINT();            /* 全割り込み受付可 */
  /* LCD にメニュー表示 */
  lcd_cursor(0,0);
  lcd_printstr(" 0:Game Start   ");
  lcd_cursor(0,1);
  lcd_printstr(" *:High Score   ");
  high_score = 0;
  while(1){
    if (key_check(10) == KEYON){     /* キーハイスコア表示*: */
      lcd_cursor(0,0);                   /* LCD にハイスコア表示 */
      lcd_printstr(" High Score is  ");
      lcd_cursor(0,1);
      lcd_printstr("                ");
      lcd_cursor(0,1);
      lcd_printstr(ntos(high_score,score_s));
```

```
    }
    if (key_check(10) == KEYOFF){     /* キーを離したらメニュー表示* */
      lcd_cursor(0,0);                  /* LCD にメッセージ表示 */
      lcd_printstr(" 0:Game Start   ");
      lcd_cursor(0,1);
      lcd_printstr(" *:High Score   ");
    }
    if (key_check(11) == KEYON){      /* キーゲームスタート0: */
      lcd_cursor(0,0);                  /* LCD に操作方法表示 */
      lcd_printstr("*:Sight  0:Trig.");
      score = game_start();             /* ゲームスタート */
      lcd_cursor(0,1);                  /* 得点表示欄のクリア */
      lcd_printstr("                ");
      if (score > high_score){          /* ハイスコアのとき */
  high_score = score;                /* ハイスコア登録 */
  lcd_cursor(0,0);                   /* ハイスコア表示 */
  lcd_printstr(" High Score !!! ");
  lcd_cursor(0,1);
  lcd_printstr(ntos(high_score,score_s));
      } else {                          /* ハイスコアでないとき*/
  lcd_cursor(0,0);                  /* スコアを表示 */
  lcd_printstr(" Your Score ... ");
  lcd_cursor(0,1);
  lcd_printstr(ntos(score,score_s));
      }
      n_time = 0;
      while (n_time < WAITFEWSEC);      /* 得点表示後にちょっと待つ */
      lcd_cursor(0,0);                  /* LCD にメッセージ表示 */
      lcd_printstr(" 0:Game Start   ");
      lcd_cursor(0,1);
      lcd_printstr(" *:High Score   ");
    }
  }
}

char *ntos(unsigned int n, char *s)
     /* 数値を文字列に変換する関数 */
     /* 引数は、整数値：(>0)n, 変換文字列が入る文字列変数ポインタ：s */
     /* 戻り値は、文字列変数ポインタ */
{
  int count, i;
  char st[MAXDIGITNUM+1];
  char d;

  count = 0;
  do {
    d = n % 10;
    n = (n - d) / 10;
    st[count] = d + '0';
    count++;
  } while (n > 0);
  for (i = 0; i < count; i++)
    s[i] = st[count - 1 - i];
  s[count] = '\0';
  return s;
}

int game_start(void)
     /* ゲームの本体UFO                                          */
     /* 戻り値: ゲームスコア                                     */
     /* 処理の高速化のためになるべく関数化しないで書いてある     */
     /* 関数を呼び出すと多くのレジスタを退避するし,               */
     /* 外部メモリのアクセスはビット幅でかつ遅い8                 */
     /* 但し, きれいに関数化しても差障りはないかもしれない       */
{
  int target, score, gameover, hit, i, j;
  int prev_key10, prev_key11, key10, key11;
  unsigned int ds;
  char t, nc;
  char disp[MAXINVNUM+1]; /* [0: 1234567890n] のような画面イメージ */
                          /* 照準区切りインベーダと ,,UFO */
  target = 0; t = '0'; score = 0;     /* 照準とスコアの初期化 */
  speed_count = 0; speed = INITSPEED; /* プレイ速度の初期化 */
  shift_flag = FALSE;                 /* 敵の前進フラグの初期化 */
```

```
159   disp[0] = '0'; disp[1] = ':';        /* 表示フィールドの初期化 */
160   for (i = 2; i < MAXINVNUM; i++) disp[i] = ' ';
161   disp[MAXINVNUM] = '\0';
162   gameover = FALSE;        /* ゲーム終了フラグの初期化 */
163   key10 = key_check(10);  /* キーバッファ照準キーの初期化() */
164   prev_key10 = KEYOFF;
165   key11 = key_check(11);  /* キーバッファ発射キーの初期化() */
166   prev_key11 = KEYOFF;
167   lcd_cursor(0,1);         /* 初期画面の表示 */
168   lcd_printstr(disp);
169
170   const int max_stage = 5;
171   int current_stage = 1;
172   short tama, teki=3, prev_teki = teki;
173
174   char info_disp[] = "S:   E:   B:    ";
175
176 //this macro cant be used as an expression
177 #define SETUP_VARIABLES()\
178   teki = prev_teki + random() % (current_stage*2);\
179   tama = teki + 99/current_stage;\
180   prev_teki = teki;\
181   target = 0;\
182   t = '0';\
183   disp[0] = '0'; disp[1] = ':';\
184   for (i = 2; i < MAXINVNUM; i++) disp[i] = ' ';\
185   disp[MAXINVNUM] = '\0';\
186   info_disp[12] = tama >= 99 ? 99 : tama / 10 + '0';  \
187   info_disp[13] = tama >= 99 ? 99 : tama % 10 + '0';    \
188   info_disp[7] = teki >= 99 ? 99 : teki / 10 + '0';   \
189   info_disp[8] = teki >= 99 ? 99 : teki % 10 + '0';\
190   info_disp[2] = current_stage / 10 + '0';\
191   info_disp[3] = current_stage % 10 + '0';\
192
193   tama = teki + teki;
194   info_disp[12] = tama / 10 + '0';
195   info_disp[13] = tama % 10 + '0';
196   info_disp[7] = teki / 10 + '0';
197   info_disp[8] = teki % 10 + '0';
198   info_disp[2] = '0';
199   info_disp[3] = '1';
200
201   lcd_cursor(0, 0);
202   lcd_printstr(info_disp);
203
204   while (gameover == FALSE){ /* ゲームオーバでなければループ */
205     /* キーの立上りを検出するための前回チェック時のキー状態を記憶 */
206     /* キーバッファ照準キー処理() */
207     if (key10 != KEYTRANS) prev_key10 = key10; /* 遷移中は前々回の値そのまま */
208     key10 = key_check(10);
209     /* キーバッファ発射キー処理() */
210     if (key11 != KEYTRANS) prev_key11 = key11; /* 遷移中は前々回の値そのまま */
211     key11 = key_check(11);
212     if ((prev_key10 == KEYOFF) && (key10 == KEYON)){
213       /* 照準キーが押されたときの処理 */
214       /* 照準は 0->1->...->8->9->n->0->... と順に変化する */
215       target++;                          /* 照準 +1 */
216       if (target > 10) target = 0;       /* の次はUFO0 */
217       if (target < 10) t = '0' + target;  /* 照準がでないときのキャラUFO */
218       else t = 'n';                      /* 照準がのときのキャラUFO */
219       disp[0] = t;                       /* 照準値を表示にセット */
220       effect('t');                       /* 効果音を鳴らす */
221       /* フィールドの表示 */
222       lcd_cursor(0,1);
223       lcd_printstr(disp);
224     }
225
226     if ((prev_key11 == KEYOFF) && (key11 == KEYON)){
227       /* 発射キーが押されたときの処理 */
228       hit = FALSE;  /* ヒット判定フラグの初期化 */
229       i = 2;        /* 最も左の敵表示位置 */
230
231       if(tama >= 1){
232   tama--;
```

```
233   info_disp[12] = tama >= 99 ? 99 : tama / 10 + '0';
234   info_disp[13] = tama >= 99 ? 99 : tama % 10 + '0';
235   while ((i < MAXINVNUM) && (hit != TRUE)){ /* 列の左から命中を探す */
236     if (disp[i] == t){    /* 照準と一致するキャラか? */
237       hit = TRUE;           /* ヒット判定   */
238       ds = MAXINVNUM - i;   /* 基本得点の計算砲台に近い程大  () */
239       score = score + ds;   /* 得点追加    */
240     if (target > 9) score = score + ds * 2;  /* なら倍の得点UFO3 */
241     for (j = i; j > 2; j--) disp[j] = disp[j-1];  /* 命中左側を右寄せ */
242     disp[2] = ' ';         /* 最も左は常に空白を詰める */
243     /* フィールドの表示 */
244     lcd_cursor(0,1);
245     lcd_printstr(disp);
246     }
247     i++;  /* 探索位置を +1 */
248   } /* ヒット判定があるか右端まで調べたら, 探索終了 */
249   if (hit == TRUE){
250     effect('s'); /* 命中時の効果音 */
251     teki--;
252     info_disp[7] = teki >= 99 ? 99 : teki / 10 + '0';
253     info_disp[8] = teki >= 99 ? 99 : teki % 10 + '0';
254     if(teki == 0){ //clear stage
255       current_stage++;
256       if(current_stage == 100){
257         gameover = TRUE;
258       }else{
259         SETUP_VARIABLES();
260         lcd_cursor(0, 0);
261         lcd_printstr("CLEAR           ");
262         wait1ms(1000);
263         lcd_cursor(0, 0);
264         lcd_printstr("NEXT STAGE      ");
265         lcd_cursor(0, 1);
266         lcd_printstr(disp);
267         wait1ms(1000);
268         lcd_cursor(0, 0);
269         lcd_printstr(info_disp);
270         continue;
271       }
272     }
273   }else{
274     effect('m');              /* 失敗時の効果音 */
275   }
276   lcd_cursor(0, 0);
277   lcd_printstr(info_disp);
278       }else{
279   effect('m');
280       }
281     }
282     /* 敵が前進するタイミングのときの処理 */
283     if (shift_flag == TRUE){           /* 前進フラグが立っているなら */
284       if (disp[2] != ' ') disp[1] = disp[2]; /* 侵略時のみ区切りに侵入 */
285       for (i = 2; i < MAXINVNUM; i++)   /* 敵全体を左につシフト1 */
286   disp[i] = disp[i + 1];
287       if (score % 10 == 1) nc = 'n';    /* スコアの最下位がなら1出現UFO */
288       else nc = (random() % 10) + '0';  /* それ以外はランダムな数字 */
289       disp[MAXINVNUM - 1] = nc;         /* 右端に新キャラを入れる */
290       shift_flag = FALSE;               /* 前進フラグの消去 */
291       lcd_cursor(0,1);                  /* フィールドの表示 */
292       lcd_printstr(disp);
293     }
294     if (disp[1] != ':') gameover = TRUE; /* 侵略されたらゲームオーバ */
295   }
296   return score; /* 得点を返す */
297 }
298
299 void effect(char c)
300      /* 効果音を発生するための関数 */
301      /* 引数: c 効果音の種類を指定 */
302 {
303   unsigned int t;
304   unsigned long p;
305
306   switch (c) {        /* 種類によって音程と音長を変えられる */
```

```
307   case 't':             /* 照準を動かしたとき */
308     t = TONE_TARGET;
309     p = LONG_TARGET;
310     break;
311   case 'm':             /* ミスしたとき */
312     t = TONE_MISS;
313     p = LONG_MISS;
314     break;
315   case 's':             /* ヒットしたとき */
316     t = TONE_HIT;
317     p = LONG_HIT;
318     break;
319   default:              /* その他のとき*/
320     t = TONE_DEF;
321     p = LONG_DEF;
322   }
323   sound_beep(t,p,32);
324 }
325
326 #pragma interrupt
327 void int_imia1(void)
328       /* キーセンス・速度調整・時間待ち用のタイマ割り込みハンドラ1 */
329 {
330   ENINT1();         /* 音程用の割り込み処理を優先させる */
331   key_sense();      /* キーセンス */
332   n_time++;         /* 時間待ち用のカウンタ +1 */
333   speed_count++;                  /* スピード調整用カウンタ +1 */
334   if (speed_count >= speed){      /* 設定値になったら */
335     speed_count = 0;              /* カウンタ初期化   */
336     speed = speed - SPEEDUP;      /* スピードアップ   */
337     shift_flag = TRUE;            /* 前進フラグ   ON */
338   }
339
340   /* 再びタイマ割り込みを使用するために必要な操作       */
341   /* タイマの割り込みフラグをクリアしないといけない  1 */
342   timer_intflag_reset(1);
343
344   ENINT();          /* を割り込み許可状態にCPU */
345 }
```

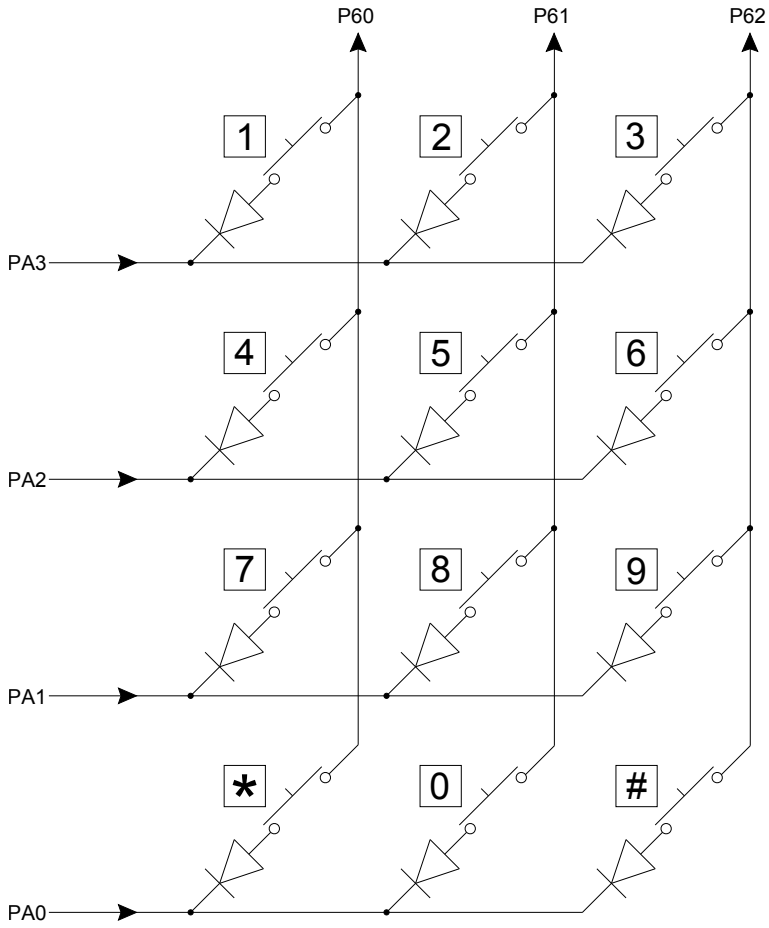

図 5: キーマトリクス回路図