

EE-208 : Rapport de projet microcontrôleur

26 mai 2025

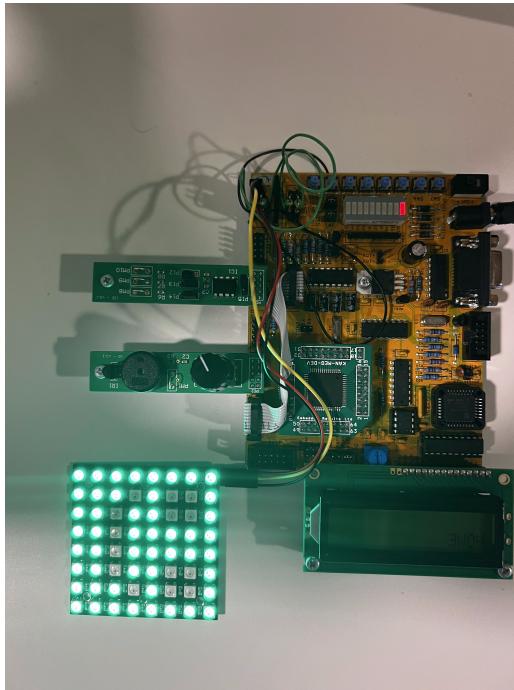


FIGURE 1 – Vue du prototype sur le Home.

1 Introduction

Ce rapport présente le développement d'un prototype médical réalisé dans le cadre du cours EE-208 : Microcontrôleurs et systèmes numériques. L'objectif du projet est de mesurer, de manière non intrusive, des paramètres médicaux chez l'enfant, et plus particulièrement chez les enfants autistes. Aujourd'hui, les dispositifs classiques peuvent être perçus comme "menaçants" par ces enfants ; notre projet propose donc une approche plus discrète et rassurante.

Dans notre prototype, nous mesurons la température à l'aide d'un capteur dédié. Pour capter l'attention et divertir l'enfant, nous utilisons ensuite la matrice de LED WS2812, capable de simuler le jeu Snake et d'afficher des animations colorées. En effet, tous les modules qui n'utilisent pas la matrice pour leur fonctionnement afficheront une image au lieu d'être éteints, afin de mieux divertir les enfants. Notre prototype n'est qu'une version simplifiée du produit final. C'est-à-dire que nous aurions aimé faire plus de jeux et d'autres éléments, mais, par contrainte de temps, nous avons décidé de bien modulariser le code et d'intégrer dans le prototype des jeux "vides". Ainsi, il ne reste plus qu'à ajouter la logique du jeu et à comprendre notre code pour développer un nouveau module.

2 Mode d'emploi

2.1 Utilisation par l'enfant (ou l'utilisateur général)

Au démarrage du prototype, un menu principal s'affiche automatiquement sur l'écran LCD. L'interface a été pensée pour être simple et intuitive, de manière à ce que l'enfant ou l'utilisateur puisse interagir facilement avec le dispositif sans accompagnement.

Quatre boutons physiques, étiquetés de **A** à **D** (cf. Figure 1), permettent la navigation dans les différents états de la machine à états finis (FSM), selon la logique suivante :

- **Bouton A (Next State)** : Permet de passer à l'état suivant dans la séquence. Par exemple, depuis l'état *Home*, une pression sur ce bouton mène successivement à *Jeu 1*, *Jeu 2*, etc., avant de revenir à *Home* dans une boucle cyclique.
- **Bouton B (Previous State)** : Permet de revenir à l'état précédent dans la séquence. Ainsi, depuis *Jeu 2*, une pression ramène à *Jeu 1*, puis à *Home*, en sens inverse.
- **Bouton C (Home State)** : Sert à revenir immédiatement à l'état *Home*, quel que soit l'état courant. Ce bouton permet à l'enfant de réinitialiser le système à tout moment.

2.2 Utilisation par le personnel médical

Le dispositif mesure la température cutanée de façon continue dès la mise en marche. Le capteur thermique reste actif en permanence, assurant une disponibilité immédiate. Lorsque le patient place ses doigts sur le capteur (signalé par une LED), la température est automatiquement mesurée et mise à jour toutes les secondes.

Le bouton D (Doctor State) permet d'entrer dans l'état *Doctor*, dédié à l'affichage de la température en temps réel sur l'écran LCD. Cet état est conçu pour un usage exclusif par le personnel médical.

Afin d'éviter toute distraction durant cette phase, **la matrice LED est automatiquement désactivée dès l'entrée dans l'état Doctor** : les jeux et animations ne sont plus affichés. Une pression sur le **bouton C** permet alors de revenir à l'état *Home*, préparant le dispositif pour un nouveau patient.

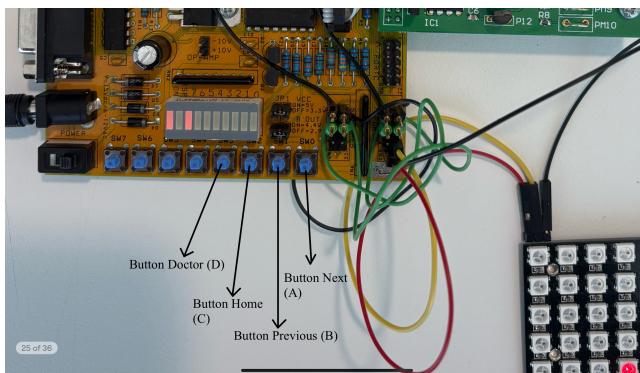


FIGURE 2 – Vue du prototype avec l'écran LCD et les boutons physiques A, B, C et D utilisés pour la navigation dans la FSM.

3 Description technique

3.1 Conception intellectuel du projet

Avant de commencer le développement, nous avons mené une phase de réflexion et de brainstorming afin de définir une idée à la fois utile, originale et réalisable dans le cadre du projet. C'est ainsi qu'est née l'idée de concevoir un prototype médical destiné à rendre la prise de température plus confortable pour les enfants autistes, en l'associant à des éléments ludiques comme des jeux interactifs.

Dès le départ, nous savions qu'il ne serait pas possible de réaliser l'intégralité du dispositif tel que nous l'imaginions dans sa version finale. Nous avons donc choisi de structurer notre projet autour d'un code **modulaire, générique et réutilisable**, de façon à pouvoir facilement l'étendre ou l'adapter ultérieurement, par exemple, en intégrant de nouveaux capteurs ou jeux.

Nous avons également accordé une grande importance à la **robustesse**, à une **structure claire** du programme et à une **documentation soignée**, afin de faciliter la compréhension, la maintenance et l'évolution du système par d'autres développeurs.

Pour mettre en oeuvre cette vision, nous avons adopté une **approche de développement hybride**, combinant les méthodes *top-down* et *bottom-up*. Nous avons commencé par implémenter certaines fonctionnalités clés, telles que la mesure de température et l'affichage sur la matrice LED, avant de structurer progressivement l'ensemble du projet autour d'une logique d'états bien définie.

Cette logique repose sur une **machine à états finis (FSM)** que nous avons conçue pour organiser les différentes phases du fonctionnement du dispositif : écran d'accueil, différents jeux, affichage médical, etc. Les transitions entre ces états sont contrôlées de manière intuitive à l'aide des boutons physiques du prototype. Cette organisation garantit la lisibilité du code, sa cohérence, et facilite l'ajout de nouvelles fonctionnalités.

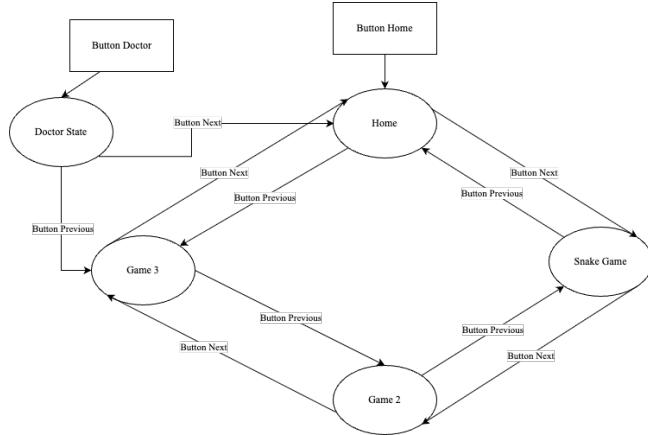


FIGURE 3 – Schéma de la FSM.

Lorsqu'on ne clique sur aucun bouton, on reste dans le même état. Notre modèle mathématique est complet et cohérent.

3.2 Connexion des ports

Voir figure ci-dessous :

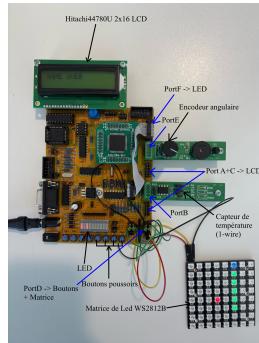


FIGURE 4 – Prototype connecté et prêt à l'usage

3.3 Description du main

Le fichier `main.asm` constitue le noyau du prototype : il orchestre l'initialisation du matériel, gère les interruptions, pilote la sonde de température et implémente la machine à états finis garantissant la modularité et la réactivité du système.

Dès le reset, une séquence d'initialisation complète prépare l'ensemble des périphériques :

- Configuration de la pile et initialisation des drivers (`LCD_init`, `wire1_init`, `encoder_init`),
- Mise en place des entrées boutons (PD0–PD3) avec résistances de tirage interne,
- Initialisation de la LED matricielle WS2812 (PD7) et du témoin " heartbeat " (PF7, actif-bas),
- Préchargement et démarrage de Timer-1 pour un tick à 1 Hz,
- Activation des interruptions externes INT0–INT3 sur front descendant.

Cette phase garantit une base robuste sur laquelle reposent toutes les fonctionnalités ultérieures.

La boucle principale se réduit à un simple commutateur sur la variable `sel` (l'état courant), qui appelle, à chaque itération, la routine d'initialisation du module correspondant :

- `home_init` pour l'écran d'accueil,
- `snake_init`, `gameTwoInit`, `gameThreeInit` pour les différents jeux,
- `doctorInit` pour l'affichage médical.

Cette logique *top-down* assure une séparation claire entre le dispatcher et les comportements spécifiques à chaque état, facilitant la lisibilité et l'évolution du code.

Le traitement de la sonde DS18B20 s'effectue de manière asynchrone : le Timer-1 ISR bascule la LED de battement et lève un drapeau `FLG_TEMP` toutes les secondes, tandis que la routine `temp_task`, appelée par chaque module d'état, gère alternativement la conversion et la lecture de la température. Ainsi, la mesure s'intègre de façon non bloquante, sans interrompre le flux principal.

Les changements d'état sont entièrement pilotés par les interruptions externes :

- INT0 : état suivant (incrément et wrap),
- INT1 : état précédent (décrément et wrap),
- INT2 : retour à l'accueil (`ST_HOME`),
- INT3 : mode "docteur" (`ST_DOCTOR`).

Ce schéma *bottom-up* permet une navigation intuitive via les boutons physiques, sans complexité dans la boucle principale.

3.4 Description du Snake game

Lorsqu'on entre dans l'état `ST_GAME1`, la routine `snake_game_init` est appelée. Elle efface l'écran, initialise l'encodeur, les données du jeu (serpent, pomme, direction) et dessine la première frame. Elle termine sur un saut vers `snake_wait`, une boucle temporelle fixe basée sur `FRAME_DELAY_MS` (500 ms).

Chaque itération appelle successivement :

- `update_game` : traite l'entrée de l'utilisateur.
- `move_snake` : calcule le déplacement suivant.
- `snake_draw` : rend le nouvel état sur la matrice.

Tant que `sel = ST_GAME1`, la boucle continue. Si le joueur perd, un message GAME OVER est affiché et le programme entre dans `freeze_game`, qui attend que `sel` change pour quitter l'état.

- `snake_body[64]` : tableau circulaire contenant les coordonnées compactées $x + 8*y$ de chaque segment du serpent. Chaque cellule est donc unique sur une grille 8x8.
- `head_idx`, `tail_idx` : indices circulaires du début et de la fin du serpent dans le tableau.
- `snake_len` : longueur actuelle du serpent (initiallement 3).
- `direction` : direction actuelle (valeurs 0 à 3 pour haut/droite/bas/gauche).
- `apple_pos` : position actuelle de la pomme, ou `0xFF` si absente.
- `turn_queue[8]`, `tq_head`, `tq_tail` : file circulaire FIFO contenant les directions validées via l'encodeur rotatif.

La routine `snake_init_data` :

- Efface tout le tableau `snake_body` en le remplissant de `EMPTY_CELL`.
- Insère les trois premiers segments du serpent aux positions 26, 27, 28.
- Fixe `head_idx = 2`, `tail_idx = 0`, `snake_len = 3`.
- Initialise la direction à droite (`DIR_RIGHTT`) et place la pomme en position 45 (5,5).
- Vide la file des directions et enregistre l'état initial de l'encodeur.

`move_snake` récupère la position de la tête actuelle, extrait `x` et `y`, et les modifie selon la direction. Si le déplacement provoque une sortie de la grille ($x < 0$ ou $x \geq 8$, etc.), on appelle `hit_wall` et le jeu est figé.

Aucun rebouclage n'est permis : toucher un bord provoque immédiatement une fin de partie (GAME OVER).

Si le serpent atteint la pomme, `snake_len` est incrémenté et la queue reste inchangée le serpent grandit. Sinon, la queue est avancée pour conserver une longueur constante. Dans tous les cas, la nouvelle position de la tête est enregistrée à `head_idx`, mis à jour de manière circulaire.

`update_game` lit la variation de l'encodeur via `encoder_update` (+1, -1 ou 0). Si une nouvelle

direction est détectée, elle est convertie en valeur `DIR_*`, puis enfilée dans `turn_queue` si elle ne provoque pas un demi-tour. Une direction est ensuite immédiatement retirée de la file et appliquée à `direction`.

La file permet de capturer plusieurs commandes rapidement saisies, et de les appliquer séquentiellement sans en perdre. Cependant, une nouvelle direction est refusée si elle est opposée à la direction actuelle (demi-tour).

La routine `place_new_apple` tente 8 fois de générer une position valide :

- Elle combine le compteur de Timer0 (`TCNT0`) et le LSB de l'ADC (`ADCL`) via un `EOR`, puis ajoute `head_idx` pour décorréler. Le registre `ADCL` ajoute juste de l'entropie et est considéré comme une source flottante.
- L'indice est converti en `x`, `y`, puis en `x+8y`.
- Ce candidat est vérifié contre tous les segments du serpent.
- Si aucune collision n'est trouvée, la position est mémorisée.

Ce mécanisme est important car il garantit un temps constant de génération (toujours 8 essais au maximum), ce qui évite les blocages en fin de partie.

Remarque : le compteur de Timer0 joue un rôle crucial ici. Il fonctionne de manière autonome et libre (sans modification de prescaler), fournissant une source de temps asynchrone qui évolue naturellement entre deux appels à `place_new_apple`. Cela rend la génération de positions pseudo-aléatoires plus variée, même sans fonction aléatoire complète.

`snake_draw` procède comme suit :

1. Efface la matrice via `matrix_solid` (noir).
2. Pour chaque segment (de `tail` à `tail + len`), décode `x`, `y`, calcule l'adresse de la LED dans le buffer (à partir de `WS_BUF_BASE`) et colorie en vert (corps) ou bleu (tête).
3. Affiche la pomme (rouge) si présente.
4. Enfin, le buffer est transmis à la matrice LED avec `ws_byte3wr`, puis "latched" via `ws_reset`.

La tête est dessinée en dernier (`tail + len - 1`) et colorée en bleu, ce qui la distingue du corps (en vert).

Si le serpent touche un mur, `hit_wall` efface le LCD et affiche GAME OVER. Le code entre alors dans une boucle `freeze_game` qui ne quitte l'état que lorsque `sel` a changé.

Ce mécanisme simple permet de laisser l'utilisateur lire l'écran de fin sans interrompre l'exécution du programme principal.

4 Interruption / Timers

Le système utilise de nombreuses **interruptions** matérielles pour assurer une opération fluide en temps réel, en particulier :

- **Interruptions externes INT0–INT3** : Associées aux quatre boutons poussoirs, elles permettent de réagir instantanément aux actions de l'utilisateur. Chaque interruption est configurée sur front descendant (*falling edge*) via le registre `EICRA` de l'ATmega128L (car nos boutons tirent la ligne vers 0 lorsqu'ils sont pressés, l'état idle étant haut grâce au pull-up). Dans la routine d'interruption correspondante, nous déterminons quel bouton a été pressé (chaque `INTx` a son vecteur) et nous mettons à jour l'état du système en entier. Par exemple, lorsque le bouton C (Home button) est cliqué, cela efface la variable qui stocke l'état, ce qui engendre un changement dans la matrice de LED et l'écran LCD. Chaque routine est soigneusement écrite en assembleur en préservant les registres utilisés (push/pop) et en effectuant un `reti` en fin. Les interruptions de boutons sont brèves : elles se limitent à changer un indicateur de sélection, le traitement complet étant effectué dans la boucle principale ou l'ISR du timer.
- **Interruption de Timer1 (comparaison)** : Le **Timer1** est configuré en mode **overflow** pour tourner en continu, quelle que soit la valeur de `sel`. Avec une horloge CPU à 4 MHz et un prescaler de 1024, il compte à 3906 Hz ; en préchargeant `TCNT1` à `0xF0BE` (soit 61630), il ne nécessite que 3906 ticks pour déborder, soit 1 s. À chaque overflow, l'ISR :

- recharge immédiatement TCNT1 à 0xF0BE pour le cycle suivant,
- bascule la LED "heartbeat" sur PF7 (active-low) afin d'indiquer visuellement l'activité système (chaque overflow du timer, chaque prise de température),
- lève le bit FLG_TEMP dans `flags`, signalant qu'une nouvelle mesure de température peut être effectuée.

Cet overflow périodique fournit une impulsion à 1 Hz et permet, en arrière-plan, de lancer ou récupérer la conversion DS18B20 via `temp_task` dès que l'état courant teste `FLG_TEMP`. Ainsi, la lecture de la température s'effectue automatiquement chaque seconde, sans bloquer la boucle principale ni dépendre du mode affiché.

5 Usage de périphérie

5.1 LCD Interface

Le fichier `lcd.asm` a été fourni par le professeur. Il constitue une bibliothèque pour piloter un écran LCD compatible **HD44780U** à l'aide du microcontrôleur **ATmega128L** fonctionnant à **4 MHz**, en utilisant l'interface **SRAM externe**.

- Le LCD est accédé via la mémoire externe :
 - 0x8000 : registre d'instruction (`LCD_IR`)
 - 0xC000 : registre de données (`LCD_DR`)
- Le fichier inclut des sous-programmes pour :
 - Écrire des instructions : `LCD_wr_ir`
 - Écrire des caractères : `LCD_wr_dr`
 - Initialiser l'écran : `LCD_init`
 - Afficher un caractère ou gérer les retours à la ligne : `LCD_putc`, `LCD_cr`, `LCD_lf`
 - Positionner le curseur : `LCD_pos`
 - Temporiser : `lcd_4us`, `lcd_2us`

Ce fichier permet de contrôler un écran LCD : affichage de texte, déplacement du curseur, configuration de l'affichage, etc. Il respecte les contraintes de temporisation du LCD, en mode 8 bits avec vérification du bit d'occupation (busy flag). Il est adapté pour un usage en environnement académique avec Atmel Studio et une carte STK300.

5.2 WS2812 LED Matrix

Le fichier `ws2812_driver.asm` fournit un pilote minimalistique pour communiquer avec une matrice 8x8 de LEDs WS2812B par bit-banging à 4 MHz. Il est conçu pour être réutilisable, compact et compatible avec des interruptions désactivées.

- La broche DATA est connectée à PORTD7. Les macros définissent :
 - `WS_PORT_REG = PORTD`, `WS_DDR_REG = DDRD`, `WS_PIN_IDX = 7`, `WS_PIN_MASK = (1 << 7)`
- Le tampon image de la matrice commence à l'adresse `WS_BUF_BASE = 0x0400`.
- Chaque pixel occupe 3 octets en ordre **GRB** : vert (`a0`), rouge (`a1`), bleu (`a2`).

Le protocole WS2812B impose un timing strictă:

- `WS_WRO` : génère un bit 0 (5 cycles 0,4µs haut / 0,85µs bas)
- `WS_WR1` : génère un bit 1 (8 cycles 0,8µs haut / 0,45µs bas)

Ces macros utilisent uniquement des instructions d'E/S synchrones avec le CPU 4MHz. Le registre `u` est utilisé comme tampon intermédiaire.

- `ws_init` : configure la broche de données en sortie.
- `ws_byte3wr` : envoie séquentiellement trois octets (G, R, B) via bit-banging. Clobber : `a0-a2, u, w`
- `ws_reset` : maintient la ligne DATA à LOW pendant au moins 50µs (commande latch des WS2812).
- `ws_idx_xy` : calcule l'index linéaire `x + 8×y` depuis `r24=x, r25=y`. Modifie `r24, u`.
- `ws_offset_idx` : convertit un index (0-63) en adresse Z pointant vers le bon triplet GRB dans le tampon. Modifie `u, w, ZL, ZH`.

L'ordre **GRB** respecte la spécification WS2812B : le vert est envoyé en premier, suivi du rouge puis du bleu.

- **WS_PUSH_ALL** : sauvegarde tous les registres susceptibles d'être altérés (**a0-a2, u, w, r24-r25, ZL, ZH**)
- **WS_POP_ALL** : restaure l'état initial dans l'ordre inverse

Utilisez-les autour de toute séquence qui appelle **ws_byte3wr**, **ws_offset_idx** ou **ws_idx_xy**, sauf si vous gérez déjà ces registres ailleurs.

Cette routine du fichier **ws2812_helpers.asm** :

1. Remplit les 64 triplets GRB du buffer **WS_BUF_BASE** avec la couleur définie par **a0, a1, a2 (G, R, B)**
2. Transmet chaque triplet via **ws_byte3wr**, protégé par **cli/sei**
3. Appelle **ws_reset** pour valider le rendu

Elle est utilisée pour effacer ou remplir rapidement l'écran avec une couleur uniforme, par exemple le noir avant un nouveau frame du jeu.

5.3 Encoder Rotatif

Le fichier **encoder.asm**, fourni par le professeur, met à disposition une bibliothèque compacte pour la lecture d'un encodeur à quadrature avec bouton poussoir sur ATmega128L à 4MHz :

- **encoder_init** : configure trois lignes de PORTE en entrée avec résistances pull-up internes.
- **encoder** : lit les transitions sur les lignes A, B (rotation) et I (bouton), met à jour deux compteurs **a0** (rotation sens horaire) et **b0** (anti-horaire), et positionne les flags Z (mouvement) et T (appui bouton).
- **encoder_update** : wrapper qui préserve l'ancienne valeur de **a0**, appelle **encoder** puis calcule **r15 = a0_new - a0_old** pour fournir directement +1/-1/0.
- Macro **CYCLIC** : aide à maintenir un compteur cyclique dans un intervalle donné.

Ces routines ne nécessitent aucune modification et s'intègrent directement au moteur de jeu pour gérer la saisie rotative et les pressions.

5.4 Temperature Sensor (DS18B20)

Le DS18B20 est un capteur de température numérique qui communique via une seule ligne de données (1-Wire). Chaque cycle de lecture se compose des étapes suivantes :

1. Réinitialisation et détection de présence
 2. **Skip ROM + Convert T**
 3. Attente $\leq 750,\text{ms}$ pour la conversion
 4. Réinitialisation et détection de présence
 5. **Skip ROM + Read Scratchpad**
- **Impulsion de réinitialisation** : Le maître force la ligne à l'état bas pendant 480 μs , relâche, puis échantillonne pendant 70 μs .
 - **Présence** : Le capteur tire la ligne à l'état bas pendant 60 à 240 μs après la relâche.
 - **Écriture dun 0** : Bas pendant 60 μs , puis relâchement pendant 10 μs .
 - **Écriture dun 1** : Bas pendant 1 μs , puis relâchement pendant 60 μs .
 - **Lecture** : Bas pendant 1 μs , relâchement, échantillonnage à 14 μs .

Le résultat sur deux octets est une valeur signée de 16 bits, exprimée en unités de $\frac{1}{16} \text{ }^{\circ}\text{C}$:

$$\text{raw} = (\text{MSB} \ll 8) | \text{LSB}, \quad T = \frac{\text{raw}}{16.0} \text{ }^{\circ}\text{C}.$$

Une interruption de minuterie toutes les secondes déclenche un changement d'état :

- **phase=0** : lance la conversion, puis passe à **phase=1**.
- **phase=1** : lit le registre **scratchpad**, puis repasse à **phase=0**.

Cela permet d'alterner entre conversion et lecture chaque seconde sans boucle d'attente active.

6 Annexe

medical_thermo/medical_thermo/definitions.asm

```
1 ; file: definitions.asm    target ATmega128L-4MHz-STK300
2 ; purpose library, definition of addresses and constants
3 ; 20171114 A.S.
4
5 ; === definitions ===
6 .nolist      ; do not include in listing
7 .set   clock  = 4000000
8
9 .def   char    = r0      ; character (ASCII)
10 .def  _sreg   = r1      ; saves the status during interrupts
11 .def  _u     = r2      ; saves working reg u during interrupt
12 .def   u      = r3      ; scratch register (macros, routines)
13
14 .def   e0    = r4      ; temporary reg for PRINTF
15 .def   e1    = r5
16
17 .equ   c    = 8
18 .def   c0   = r8      ; 8-byte register c
19 .def   c1   = r9
20 .def   c2   = r10
21 .def   c3   = r11
22
23 .equ   d    = 12      ; 4-byte register d (overlapping with c)
24 .def   d0   = r12
25 .def   d1   = r13
26 .def   d2   = r14; we use it
27 .def   d3   = r15
28
29 .def   w    = r16      ; working register for macros
30 .def   _w   = r17      ; working register for interrupts
31
32 .equ   a    = 18
33 .def   a0   = r18      ; 4-byte register a
34 .def   a1   = r19
35 .def   a2   = r20
36 .def   a3   = r21
37
38 .equ   b    = 22
39 .def   b0   = r22      ; 4-byte register b
40 .def   b1   = r23
41 .def   b2   = r24
42 .def   b3   = r25
43
44 .equ   px   = 26      ; pointer x
45 .equ   py   = 28      ; pointer y
46 .equ   pz   = 30      ; pointer z
47
48 ; === ASCII codes
49 .equ   BEL = 0x07    ; bell
```

```

50 .equ HT =0x09 ; horizontal tab
51 .equ TAB =0x09 ; tab
52 .equ LF =0x0a ; line feed
53 .equ VT =0x0b ; vertical tab
54 .equ FF =0x0c ; form feed
55 .equ CR =0x0d ; carriage return
56 .equ SPACE =0x20 ; space code
57 .equ DEL =0x7f ; delete
58 .equ BS =0x08 ; back space
59
60 ; === STK-300 ===
61 .equ LED = PORTB ; LEDs on STK-300
62 .equ BUTTON = PIND ; buttons on the STK-300
63
64 ; === module M2 (encoder/speaker/IR remote) ===
65 .equ SPEAKER = 2 ; piezo speaker
66 .equ ENCOD_A = 4 ; angular encoder A
67 .equ ENCOD_B = 5 ; angular encoder B
68 .equ ENCOD_I = 6 ; angular encoder button
69 .equ IR = 7 ; IR module for PCM remote control system
70
71 ; === module M5 (I2C/1Wire) ===
72 .equ SCL = 0 ; I2C serial clock
73 .equ SDA = 1 ; I2C serial data
74 .equ DQ = 5 ; Dallas 1Wire
75 ; master transmitter status codes, Table 88
76 .equ I2CMT_START = 0x08 ; start
77 .equ I2CMT_REPSTART = 0x10 ; repeated start
78 .equ I2CMT_SLA_ACK= 0x18 ; slave ack
79 .equ I2CMT_SLA_NOACK = 0x20 ; slave no ack
80 .equ I2CMT_DATA_ACK = 0x28 ; data write, ack
81 .equ I2CMT_DATA_NOACK = 0x30 ; data write, no ack
82 ; master receiver status codes, Table 89
83 .equ I2CMR_SLA_ACK = 0x40 ; slave address ack
84 .equ I2CMR_SLA_NACK = 0x48 ; slave address no ack
85 .equ I2CMR_DATA_ACK = 0x50 ; master data ack
86 .equ I2CMR_DATA_NACK= 0x58 ; master data no ack
87
88 ; === module M4 (Keyboard/Sharp/Servo) ===
89 .equ KB_CLK = 0 ; PC-AT keyboard clock line
90 .equ KB_DAT = 1 ; PC-AT keyboard data line
91 .equ GP2_CLK = 2 ; Sharp GP2D02 distance measuring sensor
92 .equ GP2_DAT = 3 ; Sharp GP2D02 distance measuring sensor
93 .equ GP2_AVAL = 3; Shart GP2Y0A21 distance measuring sensor
94 .equ SERV01 = 4 ; Futaba position servo
95
96 ; === module M3 (potentiometer/BNC) ===
97 .equ POT = 0 ; potentiometer
98 .equ BNC1 = 2 ; BNC input
99 .equ BNC2 = 4 ; BNC input
100 .list

```

101 |

medical_thermo/medical_thermo/macros.asm

```
1 ; file: macros.asm    target ATmega128L-4MHz-STK300
2 ; purpose library, general-purpose macros
3 ; author (c) R.Holzer (adapted MICR0210/EE208 A.Schmid)
4 ; v2019.01 20180820 Axs
5
6 ; =====
7 ;   pointers
8 ; =====
9
10; --- loading an immediate into a pointer XYZ,SP ---
11.macro LDIX    ; sram
12  ldi xl, low(@0)
13  ldi xh,high(@0)
14  .endmacro
15.macro LDIY    ; sram
16  ldi yl, low(@0)
17  ldi yh,high(@0)
18  .endmacro
19.macro LDIZ    ; sram
20  ldi zl, low(@0)
21  ldi zh,high(@0)
22
23  .endmacro
24.macro LDZD    ; sram, reg ; sram+reg -> Z
25  mov zl,@1
26  clr zh
27  subi zl, low(-@0)
28  sbci zh,high(-@0)
29  .endmacro
30.macro LDSP    ; sram
31  ldi r16, low(@0)
32  out spl,r16
33  ldi r16,high(@0)
34  out sph,r16
35  .endmacro
36
37; --- load/store SRAM addr into pointer XYZ ---
38.macro LDSX    ; sram
39  lds xl,@0
40  lds xh,@0+1
41  .endmacro
42.macro LDSY    ; sram
43  lds yl,@0
44  lds yh,@0+1
45  .endmacro
46.macro LDSZ    ; sram
47  lds zl,@0
48  lds zh,@0+1
49  .endmacro
```

```

50 .macro STSX      ; sram
51   sts @0, xl
52   sts @0+1, xh
53   .endmacro
54 .macro STSY      ; sram
55   sts @0, yl
56   sts @0+1, yh
57   .endmacro
58 .macro STSZ      ; sram
59   sts @0, zl
60   sts @0+1, zh
61   .endmacro
62
63 ; --- push/pop pointer XYZ ---
64 .macro PUSHX          ; push X
65   push  xl
66   push  xh
67   .endmacro
68 .macro POPX          ; pop X
69   pop   xh
70   pop   xl
71   .endmacro
72
73 .macro PUSHY          ; push Y
74   push  yl
75   push  yh
76   .endmacro
77 .macro POPY          ; pop Y
78   pop   yh
79   pop   yl
80   .endmacro
81
82 .macro PUSHZ          ; push Z
83   push  zl
84   push  zh
85   .endmacro
86 .macro POPZ          ; pop Z
87   pop   zh
88   pop   zl
89   .endmacro
90
91 ; --- multiply/divide Z ---
92 .macro MUL2Z          ; multiply Z by 2
93   lsl  zl
94   rol  zh
95   .endmacro
96 .macro DIV2Z          ; divide Z by 2
97   lsr  zh
98   ror  zl
99   .endmacro
100

```

```

101 ; --- add register to pointer XYZ ---
102 .macro ADDX    ;reg          ; x <- y+reg
103   add xl,@0
104   brcc PC+2
105   subi xh,-1      ; add carry
106 .endmacro
107 .macro ADDY    ;reg          ; y <- y+reg
108   add yl,@0
109   brcc PC+2
110   subi yh,-1      ; add carry
111 .endmacro
112 .macro ADDZ    ;reg          ; z <- z+reg
113   add zl,@0
114   brcc PC+2
115   subi zh,-1      ; add carry
116 .endmacro
117
118 ; =====
119 ;   miscellaneous
120 ; =====
121
122 ; --- output/store (regular I/O space) immediate value ---
123 .macro OUTI    ; port,k    output immediate value to port
124   ldi w,@1
125   out @0,w
126 .endmacro
127
128 ; --- output/store (extended I/O space) immediate value ---
129 .macro OUTEI   ; port,k    output immediate value to port
130   ldi w,@1
131   sts @0,w
132 .endmacro
133
134 ; --- add immediate value ---
135 .macro ADDI
136   subi @0,-@1
137 .endmacro
138 .macro ADCI
139   sbci @0,-@1
140 .endmacro
141
142 ; --- inc/dec with range limitation ---
143 .macro INC_LIM ; reg,limit
144   cpi @0,@1
145   brlo PC+3
146   ldi @0,@1
147   rjmp PC+2
148   inc @0
149 .endmacro
150
151 .macro DEC_LIM ; reg,limit

```

```

152    cpi @0,@1
153    breq    PC+5
154    brlo    PC+3
155    dec @0
156    rjmp   PC+2
157    ldi @0,@1
158    .endmacro
159
160; --- inc/dec with cyclic range ---
161.macro INC_CYC ; reg,low,high
162    cpi @0,@2
163    brsh    _low     ; reg>=high then reg=low
164    cpi @0,@1
165    brlo    _low     ; reg< low  then reg=low
166    inc @0
167    rjmp   _done
168 _low:  ldi @0,@1
169 _done:
170    .endmacro
171
172.macro DEC_CYC ; reg,low,high
173    cpi @0,@1
174    breq    _high    ; reg=low then reg=high
175    brlo    _high    ; reg<low then reg=high
176    dec @0
177    cpi @0,@2
178    brsh    _high    ; reg>=high then high
179    rjmp   _done
180 _high: ldi @0,@2
181 _done:
182    .endmacro
183
184.macro INCDEC  ;port,b1,b2,reg,low,high
185    sbic   @0,@1
186    rjmp   PC+6
187
188    cpi @3,@5
189    brlo    PC+3
190    ldi @3,@4
191    rjmp   PC+2
192    inc @3
193
194    sbic   @0,@2
195    rjmp   PC+7
196
197    cpi @3,@4
198    breq   PC+5
199    brlo    PC+3
200    dec @3
201    rjmp   PC+2
202    ldi @3,@5

```

```

203     .endmacro
204
205 ; --- wait loops ---
206 ; wait 10...196608 cycles
207 .macro WAIT_C ; k
208     ldi w, low(@0-7)/3
209     mov u,w          ; u=LSB
210     ldi w,high(@0-7)/3+1 ; w=MSB
211     dec u
212     brne    PC-1
213     dec u
214     dec w
215     brne    PC-4
216     .endmacro
217
218 ; wait micro-seconds (us)
219 ; us = x*3*1000'000/clock) ==> x=us*clock/3000'000
220 .macro WAIT_US ; k
221     ldi w, low(clock/1000*@0/3000)-1
222     mov u,w
223     ldi w,high((clock/1000*@0/3000)-1)+1 ; set up: 3 cycles
224     dec u
225     brne    PC-1          ; inner loop: 3 cycles
226     dec u      ; adjustment for outer loop
227     dec w
228     brne    PC-4
229     .endmacro
230
231 ; wait mili-seconds (ms)
232 .macro WAIT_MS ; k
233     ldi w, low(@0)
234     mov u,w      ; u = LSB
235     ldi w,high(@0)+1    ; w = MSB
236 wait_ms:
237     push   w      ; wait 1000 usec
238     push   u
239     ldi w, low(clock/3000)-5
240     mov u,w
241     ldi w,high((clock/3000)-5)+1
242     dec u
243     brne    PC-1          ; inner loop: 3 cycles
244     dec u      ; adjustment for outer loop
245     dec w
246     brne    PC-4
247     pop    u
248     pop    w
249
250     dec u
251     brne    wait_ms
252     dec w
253     brne    wait_ms

```

```

254     .endmacro
255
256 ; --- conditional jumps/calls ---
257 .macro JC0          ; jump if carry=0
258     brcs    PC+2
259     rjmp    @0
260     .endmacro
261 .macro JC1          ; jump if carry=1
262     brcc    PC+2
263     rjmp    @0
264     .endmacro
265
266 .macro JK ; reg,k,addr      ; jump if reg=k
267     cpi @0,@1
268     breq   @2
269     .endmacro
270 .macro _JK ; reg,k,addr      ; jump if reg=k
271     cpi @0,@1
272     brne   PC+2
273     rjmp   @2
274     .endmacro
275 .macro JNK ; reg,k,addr      ; jump if not(reg=k)
276     cpi @0,@1
277     brne   @2
278     .endmacro
279
280 .macro CK ; reg,k,addr      ; call if reg=k
281     cpi @0,@1
282     brne   PC+2
283     rcall  @2
284     .endmacro
285 .macro CNK ; reg,k,addr      ; call if not(reg=k)
286     cpi @0,@1
287     breq   PC+2
288     rcall  @2
289     .endmacro
290
291 .macro JSK ; sram,k,addr      ; jump if sram=k
292     lds w,@0
293     cpi w,@1
294     breq   @2
295     .endmacro
296 .macro JSNK      ; sram,k,addr      ; jump if not(sram=k)
297     lds w,@0
298     cpi w,@1
299     brne   @2
300     .endmacro
301
302 ; --- loops ---
303 .macro DJNZ      ; reg,addr      ; decr and jump if not zero
304     dec @0

```

```

305     brne    @1
306     .endmacro
307 .macro DJNK    ; reg,k,addr      ; decr and jump if not k
308     dec @0
309     cpi @0,@1
310     brne    @2
311     .endmacro
312
313 .macro IJNZ    ; reg,addr   ; inc and jump if not zero
314     inc @0
315     brne    @1
316     .endmacro
317 .macro IJNK    ; reg,k,addr      ; inc and jump if not k
318     inc @0
319     cpi @0,@1
320     brne    @2
321     .endmacro
322 .macro _IJNK    ; reg,k,addr      ; inc and jump if not k
323     inc @0
324     ldi w,@1
325     cp @0,w
326     brne    @2
327     .endmacro
328
329 .macro ISJNK    ; sram,k,addr   ; inc sram and jump if not k
330     lds w,@0
331     inc w
332     sts @0,w
333     cpi w,@1
334     brne    @2
335     .endmacro
336 .macro _ISJNK   ; sram,k,addr   ; inc sram and jump if not k
337     lds w,@0
338     inc w
339     sts @0,w
340     cpi w,@1
341     breq    PC+2
342     rjmp    @2
343     .endmacro
344
345 .macro DSJNK    ; sram,k,addr   ; dec sram and jump if not k
346     lds w,@0
347     dec w
348     sts @0,w
349     cpi w,@1
350     brne    @2
351     .endmacro
352
353 ; --- table lookup ---
354 .macro LOOKUP   ;reg, index,tbl
355     push    ZL

```

```

356    push    ZH
357    mov zl,@1      ; move index into z
358    clr zh
359    subi   zl, low(-2*@2) ; add base address of table
360    sbci   zh,high(-2*@2)
361    lpm      ; load program memory (into r0)
362    mov @0,r0
363    pop ZH
364    pop ZL
365    .endmacro
366
367 .macro L00KUP2 ;r1,r0, index,tbl
368    mov zl,@2      ; move index into z
369    clr zh
370    lsl zl      ; multiply by 2
371    rol zh
372    subi   zl, low(-2*@3) ; add base address of table
373    sbci   zh,high(-2*@3)
374    lpm      ; get LSB byte
375    mov w,r0      ; temporary store LSB in w
376    adiw   zl,1      ; increment Z
377    lpm      ; get MSB byte
378    mov @0,r0      ; mov MSB to res1
379    mov @1,w      ; mov LSB to res0
380    .endmacro
381
382 .macro L00KUP4 ;r3,r2,r1,r0, index,tbl
383    mov zl,@4      ; move index into z
384    clr zh
385    lsl zl      ; multiply by 2
386    rol zh
387    lsl zl      ; multiply by 2
388    rol zh
389    subi   zl, low(-2*@5) ; add base address of table
390    sbci   zh,high(-2*@5)
391    lpm
392    mov @1,r0      ; load high word LSB
393    adiw   zl,1
394    lpm
395    mov @0,r0      ; load high word MSB
396    adiw   zl,1
397    lpm
398    mov @3,r0      ; load low word LSB
399    adiw   zl,1
400    lpm
401    mov @2,r0      ; load low word MSB
402    .endmacro
403
404 .macro LOOKDOWN ;reg,index,tbl
405    ldi ZL, low(2*@2) ; load table address
406    ldi ZH,high(2*@2)

```

```

407     clr @1
408 loop:   lpm
409     cp r0,@0
410     breq    found
411     inc @1
412     adiw    ZL,1
413     tst r0
414     breq    notfound
415     rjmp    loop
416 notfound:
417     ldi @1,-1
418 found:
419     .endmacro
420
421 ; --- branch table ---
422 .macro C_TBL ; reg,tbl
423     ldi ZL, low(2*@1)
424     ldi ZH,high(2*@1)
425     lsl @0
426     add ZL,@0
427     brcc PC+2
428     inc ZH
429     lpm
430     push r0
431     lpm
432     mov zh,r0
433     pop zl
434     icall
435     .endmacro
436 .macro J_TBL ; reg,tbl
437     ldi ZL, low(2*@1)
438     ldi ZH,high(2*@1)
439     lsl @0
440     add ZL,@0
441     brcc PC+2
442     inc ZH
443     lpm
444     push r0
445     lpm
446     mov zh,r0
447     pop zl
448     ijmp
449     .endmacro
450
451 .macro BRANCH ; reg           ; branching using the stack
452     ldi w, low(tbl)
453     add w,@0
454     push w
455     ldi w,high(tbl)
456     brcc PC+2
457     inc w

```

```

458     push    w
459     ret
460 tbl:
461     .endmacro
462
463 ; --- multiply/division ---
464 .macro DIV2      ; reg
465     lsr @0
466     .endmacro
467 .macro DIV4      ; reg
468     lsr @0
469     lsr @0
470     .endmacro
471 .macro DIV8      ; reg
472     lsr @0
473     lsr @0
474     lsr @0
475     .endmacro
476
477 .macro MUL2      ; reg
478     lsl @0
479     .endmacro
480 .macro MUL4      ; reg
481     lsl @0
482     lsl @0
483     .endmacro
484 .macro MUL8      ; reg
485     lsl @0
486     lsl @0
487     lsl @0
488     .endmacro
489
490 ; =====
491 ;   extending existing instructions
492 ; =====
493
494 ; --- immediate ops with r0..r15 ---
495 .macro _ADDI
496     ldi w,@1
497     add @0,w
498     .endmacro
499 .macro _ADCI
500     ldi w,@1
501     adc @0,w
502     .endmacro
503 .macro _SUBI
504     ldi w,@1
505     sub @0,w
506     .endmacro
507 .macro _SBCI
508     ldi w,@1

```

```

509      sbc @0,w
510      .endmacro
511 .macro _ANDI
512     ldi w,@1
513     and @0,w
514     .endmacro
515 .macro _ORI
516     ldi w,@1
517     or  @0,w
518     .endmacro
519 .macro _EORI
520     ldi w,@1
521     eor @0,w
522     .endmacro
523 .macro _SBR
524     ldi w,@1
525     or  @0,w
526     .endmacro
527 .macro _CBR
528     ldi w,~@1
529     and @0,w
530     .endmacro
531 .macro _CPI
532     ldi w,@1
533     cp   @0,w
534     .endmacro
535 .macro _LDI
536     ldi w,@1
537     mov @0,w
538     .endmacro
539
540 ; --- bit access for port p32..p63 ---
541 .macro _SBI
542     in  w,@0
543     ori w,1<<@1
544     out @0,w
545     .endmacro
546 .macro _CBI
547     in  w,@0
548     andi w,~(1<<@1)
549     out @0,w
550     .endmacro
551
552 ; --- extending branch distance to +/-2k ---
553 .macro _BREQ
554     brne PC+2
555     rjmp @0
556     .endmacro
557 .macro _BRNE
558     breq PC+2
559     rjmp @0

```

```
560     .endmacro
561 .macro _BRCS
562     brcc    PC+2
563     rjmp    @0
564     .endmacro
565 .macro _BRCC
566     brcs    PC+2
567     rjmp    @0
568     .endmacro
569 .macro _BRSH
570     brlo    PC+2
571     rjmp    @0
572     .endmacro
573 .macro _BRL0
574     brsh    PC+2
575     rjmp    @0
576     .endmacro
577 .macro _BRMI
578     brpl    PC+2
579     rjmp    @0
580     .endmacro
581 .macro _BRPL
582     brmi    PC+2
583     rjmp    @0
584     .endmacro
585 .macro _BRGE
586     brlt    PC+2
587     rjmp    @0
588     .endmacro
589 .macro _BRLT
590     brge    PC+2
591     rjmp    @0
592     .endmacro
593 .macro _BRHS
594     brhc    PC+2
595     rjmp    @0
596     .endmacro
597 .macro _BRHC
598     brhs    PC+2
599     rjmp    @0
600     .endmacro
601 .macro _BRTS
602     brtc    PC+2
603     rjmp    @0
604     .endmacro
605 .macro _BRTC
606     brts    PC+2
607     rjmp    @0
608     .endmacro
609 .macro _BRVS
610     brvc    PC+2
```

```

611     rjmp    @0
612     .endmacro
613 .macro _BRVC
614     brvs    PC+2
615     rjmp    @0
616     .endmacro
617 .macro _BRIE
618     brid    PC+2
619     rjmp    @0
620     .endmacro
621 .macro _BRID
622     brie    PC+2
623     rjmp    @0
624     .endmacro
625
626 ; =====
627 ; bit operations
628 ; =====
629
630 ; --- moving bits ---
631 .macro MOVB    ; reg1,b1, reg2,b2 ; reg1,bit1 <- reg2,bit2
632     bst @2,@3
633     bld @0,@1
634     .endmacro
635 .macro OUTB    ; port1,b1, reg2,b2 ; port1,bit1 <- reg2,bit2
636     sbrs @2,@3
637     cbi @0,@1
638     sbrc @2,@3
639     sbi @0,@1
640     .endmacro
641 .macro INB ; reg1,b1, port2,b2 ; reg1,bit1 <- port2,bit2
642     sbis @2,@3
643     cbr @0,1<<@1
644     sbic @2,@3
645     sbr @0,1<<@1
646     .endmacro
647
648 .macro Z2C          ; zero to carry
649     sec
650     breq   PC+2      ; (Z=1)
651     clc
652     .endmacro
653 .macro Z2INVC         ; zero to inverse carry
654     sec
655     brne   PC+2      ; (Z=0)
656     clc
657     .endmacro
658
659 .macro C2Z          ; carry to zero
660     sez
661     brcs   PC+2      ; (C=1)

```

```

662     clz
663     .endmacro
664
665     .macro B2C ; reg,b          ; bit to carry
666         sbrc    @0,@1
667         sec
668         sbrs    @0,@1
669         clc
670     .endmacro
671     .macro C2B ; reg,b          ; carry to bit
672         brcc    PC+2
673         sbr @0,(1<<@1)
674         brcs    PC+2
675         cbr @0,(1<<@1)
676     .endmacro
677     .macro P2C ; port,b          ; port to carry
678         sbic    @0,@1
679         sec
680         sbis    @0,@1
681         clc
682     .endmacro
683     .macro C2P ; port,b          ; carry to port
684         brcc    PC+2
685         sbi @0,@1
686         brcs    PC+2
687         cbi @0,@1
688     .endmacro
689
690 ; --- inverting bits ---
691     .macro INVB   ; reg,bit      ; inverse reg,bit
692         ldi w,(1<<@1)
693         eor @0,w
694     .endmacro
695     .macro INVP   ; port,bit      ; inverse port,bit
696         sbis    @0,@1
697         rjmp   PC+3
698         cbi @0,@1
699         rjmp   PC+2
700         sbi @0,@1
701     .endmacro
702     .macro INVC           ; inverse carry
703         brcs    PC+3
704         sec
705         rjmp   PC+2
706         clc
707     .endmacro
708
709 ; --- setting a single bit ---
710     .macro SETBIT  ; reg(0..7)
711 ; in    reg (0..7)

```

```

712 ; out    reg with bit (0..7) set to 1.
713 ; 0=00000001
714 ; 1=00000010
715 ; ...
716 ; 7=10000000
717     mov w,@0
718     clr @0
719     inc @0
720     andi   w,0b111
721     breq   PC+4
722     lsl @0
723     dec w
724     brne   PC-2
725     .endmacro
726
727 ; --- logical operations with masks ---
728 .macro MOVMSK ; reg1,reg2,mask      ; reg1 <- reg2 (mask)
729     ldi w,~@2
730     and @0,w
731     ldi w,@2
732     and @1,w
733     or  @0,@1
734     .endmacro
735 .macro ANDMSK ; reg1,reg2,mask      ; reg1 <- ret 1 AND reg2 (mask)
736     mov w,@1
737     ori w,~@2
738     and @0,w
739     .endmacro
740 .macro ORMSK  ; reg1,reg2,mask      ; reg1 <- ret 1 AND reg2 (mask)
741     mov w,@1
742     andi   w,@2
743     or  @0,w
744     .endmacro
745
746 ; --- logical operations on bits ---
747 .macro ANDB    ; r1,b1, r2,b2, r3,b3  ; reg1,b1 <- reg2,b2 AND reg3,b3
748     set
749     sbrs   @4,@5
750     clt
751     sbrs   @2,@3
752     clt
753     bld @0,@1
754     .endmacro
755 .macro ORB ; r1,b1, r2,b2, r3,b3  ; reg1.b1 <- reg2.b2 OR reg3.b3
756     clt
757     sbrc   @4,@5
758     set
759     sbrc   @2,@3
760     set
761     bld @0,@1
762     .endmacro

```

```

763 .macro EORB    ; r1,b1, r2,b2, r3,b3    ; reg1.b1 <- reg2.b2 XOR reg3.b3
764     sbrc    @4,@5
765     rjmp    f1
766 f0: bst @2,@3
767     rjmp    PC+4
768 f1: set
769     sbrc    @0,@1
770     clt
771     bld @0,@0
772     .endmacro
773
774 ; --- operations based on register bits ---
775 .macro FB0 ; reg,bit        ; bit=0
776     cbr @0,1<<@1
777     .endmacro
778 .macro FB1 ; reg,bit        ; bit=1
779     sbr @0,1<<@1
780     .endmacro
781 .macro _FB0    ; reg,bit        ; bit=0
782     ldi w,~(1<<@1)
783     and @0,w
784     .endmacro
785 .macro _FB1    ; reg,bit        ; bit=1
786     ldi w,1<<@1
787     or  @0,w
788     .endmacro
789 .macro SB0 ; reg,bit,addr    ; skip if bit=0
790     sbrc    @0,@1
791     .endmacro
792 .macro SB1 ; reg,bit,addr    ; skip if bit=1
793     sbrs    @0,@1
794     .endmacro
795 .macro JB0 ; reg,bit,addr    ; jump if bit=0
796     sbrs    @0,@1
797     rjmp    @2
798     .endmacro
799 .macro JB1 ; reg,bit,addr    ; jump if bit=1
800     sbrc    @0,@1
801     rjmp    @2
802     .endmacro
803 .macro CB0 ; reg,bit,addr    ; call if bit=0
804     sbrs    @0,@1
805     rcall   @2
806     .endmacro
807 .macro CB1 ; reg,bit,addr    ; call if bit=1
808     sbrc    @0,@1
809     rcall   @2
810     .endmacro
811 .macro WB0 ; reg,bit        ; wait if bit=0
812     sbrs    @0,@1

```

```

813    rjmp   PC-1
814    .endmacro
815 .macro WB1 ; reg,bit      ; wait if bit=1
816     sbrc   @0,@1
817     rjmp   PC-1
818    .endmacro
819 .macro RB0 ; reg,bit      ; return if bit=0
820     sbrs   @0,@1
821     ret
822    .endmacro
823 .macro RB1 ; reg,bit      ; return if bit=1
824     sbrc   @0,@1
825     ret
826    .endmacro
827
828 ; wait if bit=0 with timeout
829 ; if timeout (in units of 5 cyc) then jump to addr
830 .macro WB0T  ; reg,bit,timeout,addr
831     ldi w,@2+1
832     dec w   ; 1 cyc
833     breq   @3  ; 1 cyc
834     sbrs   @0,@1  ; 1 cyc
835     rjmp   PC-3  ; 2 cyc = 5 cycles
836    .endmacro
837
838 ; wait if bit=1 with timeout
839 ; if timeout (in units of 5 cyc) then jump to addr
840 .macro WB1T  ; reg,bit,timeout,addr
841     ldi w,@2+1
842     dec w   ; 1 cyc
843     breq   @3  ; 1 cyc
844     sbrc   @0,@1  ; 1 cyc
845     rjmp   PC-3  ; 2 cyc = 5 cycles
846    .endmacro
847
848 ; --- operations based on port bits ---
849 .macro P0  ; port,bit      ; port=0
850     cbi @0,@1
851    .endmacro
852 .macro P1  ; port,bit      ; port=1
853     sbi @0,@1
854    .endmacro
855 .macro SP0 ; port,bit      ; skip if port=0
856     sbic   @0,@1
857    .endmacro
858 .macro SP1 ; port,bit      ; skip if port=1
859     sbis   @0,@1
860    .endmacro
861 .macro JP0 ; port,bit,addr  ; jump if port=0
862     sbis   @0,@1
863     rjmp   @2

```

```

864     .endmacro
865 .macro JP1 ; port,bit,addr      ; jump if port=1
866     sbic  @0,@1
867     rjmp  @2
868     .endmacro
869 .macro CP0 ; port,bit,addr      ; call if port=0
870     sbis  @0,@1
871     rcall @2
872     .endmacro
873 .macro CP1 ; port,bit,addr      ; call if port=1
874     sbic  @0,@1
875     rcall @2
876     .endmacro
877 .macro WP0 ; port,bit      ; wait if port=0
878     sbis  @0,@1
879     rjmp  PC-1
880     .endmacro
881 .macro WP1 ; port,bit      ; wait if port=1
882     sbic  @0,@1
883     rjmp  PC-1
884     .endmacro
885 .macro RP0 ; port,bit      ; return if port=0
886     sbis  @0,@1
887     ret
888     .endmacro
889 .macro RP1 ; port,bit      ; return if port=1
890     sbic  @0,@1
891     ret
892     .endmacro
893
894 ; wait if port=0 with timeout
895 ; if timeout (in units of 5 cyc) then jump to addr
896 .macro WP0T ; port,bit,timeout,addr
897     ldi w,@2+1
898     dec w ; 1 cyc
899     breq @3 ; 1 cyc
900     sbis @0,@1 ; 1 cyc
901     rjmp PC-3 ; 2 cyc = 5 cycles
902     .endmacro
903
904 ; wait if port=1 with timeout
905 ; if timeout (in units of 5 cyc) then jump to addr
906 .macro WP1T ; port,bit,timeout,addr
907     ldi w,@2+1
908     dec w ; 1 cyc
909     breq @3 ; 1 cyc
910     sbic @0,@1 ; 1 cyc
911     rjmp PC-3 ; 2 cyc = 5 cycles
912     .endmacro
913

```

```

914 ; =====
915 ;   multi-byte operations
916 ; =====
917
918 .macro SWAP4          ; swap 2 variables
919   mov w ,@0
920   mov @0,@4
921   mov @4,w
922   mov w ,@1
923   mov @1,@5
924   mov @5,w
925   mov w ,@2
926   mov @2,@6
927   mov @6,w
928   mov w ,@3
929   mov @3,@7
930   mov @7,w
931   .endmacro
932 .macro SWAP3
933   mov w ,@0
934   mov @0,@3
935   mov @3,w
936   mov w ,@1
937   mov @1,@4
938   mov @4,w
939   mov w ,@2
940   mov @2,@5
941   mov @5,w
942   .endmacro
943 .macro SWAP2
944   mov w ,@0
945   mov @0,@2
946   mov @2,w
947   mov w ,@1
948   mov @1,@3
949   mov @3,w
950   .endmacro
951 .macro SWAP1
952   mov w ,@0
953   mov @0,@1
954   mov @1,w
955   .endmacro
956
957 .macro LDX4    ; r..r0      ; load from (x+)
958   ld @3,x+
959   ld @2,x+
960   ld @1,x+
961   ld @0,x+
962   .endmacro
963 .macro LDX3    ; r..r0
964   ld @2,x+

```

```

965    ld @1,x+
966    ld @0,x+
967    .endmacro
968 .macro LDX2 ;r..r0
969    ld @1,x+
970    ld @0,x+
971    .endmacro
972
973 .macro LDY4 ;r..r0      ; load from (y+)
974    ld @3,y+
975    ld @2,y+
976    ld @1,y+
977    ld @0,y+
978    .endmacro
979 .macro LDY3 ;r..r0
980    ld @2,y+
981    ld @1,y+
982    ld @0,y+
983    .endmacro
984 .macro LDY2 ;r..r0
985    ld @1,y+
986    ld @0,y+
987    .endmacro
988
989 .macro LDZ4 ;r..r0      ; load from (z+)
990    ld @3,z+
991    ld @2,z+
992    ld @1,z+
993    ld @0,z+
994    .endmacro
995 .macro LDZ3 ;r..r0
996    ld @2,z+
997    ld @1,z+
998    ld @0,z+
999    .endmacro
1000 .macro LDZ2 ;r..r0
1001    ld @1,z+
1002    ld @0,z+
1003    .endmacro
1004
1005 .macro STX4 ;r..r0      ; store to (x+)
1006    st x+,@3
1007    st x+,@2
1008    st x+,@1
1009    st x+,@0
1010    .endmacro
1011 .macro STX3 ;r..r0
1012    st x+,@2
1013    st x+,@1
1014    st x+,@0

```

```

1015     .endmacro
1016 .macro STX2      ;r..r0
1017     st  x+,@1
1018     st  x+,@0
1019     .endmacro
1020
1021 .macro STY4      ;r..r0      ; store to (y+)
1022     st  y+,@3
1023     st  y+,@2
1024     st  y+,@1
1025     st  y+,@0
1026     .endmacro
1027 .macro STY3      ;r..r0
1028     st  y+,@2
1029     st  y+,@1
1030     st  y+,@0
1031     .endmacro
1032 .macro STY2      ;r..r0
1033     st  y+,@1
1034     st  y+,@0
1035     .endmacro
1036
1037 .macro STZ4      ;r..r0      ; store to (z+)
1038     st  z+,@3
1039     st  z+,@2
1040     st  z+,@1
1041     st  z+,@0
1042     .endmacro
1043 .macro STZ3      ;r..r0
1044     st  z+,@2
1045     st  z+,@1
1046     st  z+,@0
1047     .endmacro
1048 .macro STZ2      ;r..r0
1049     st  z+,@1
1050     st  z+,@0
1051     .endmacro
1052
1053 .macro STI4      ;addr,k      ; store immediate
1054     ldi w, low(@1)
1055     sts @0+0,w
1056     ldi w, high(@1)
1057     sts @0+1,w
1058     ldi w,byte3(@1)
1059     sts @0+2,w
1060     ldi w,byte4(@1)
1061     sts @0+3,w
1062     .endmacro
1063 .macro STI3      ;addr,k
1064     ldi w, low(@1)
1065     sts @0+0,w

```

```

1066    ldi w, high(@1)
1067    sts @0+1,w
1068    ldi w,byte3(@1)
1069    sts @0+2,w
1070    .endmacro
1071 .macro STI2      ;addr,k
1072     ldi w, low(@1)
1073     sts @0+0,w
1074     ldi w, high(@1)
1075     sts @0+1,w
1076     .endmacro
1077 .macro STI ;addr,k
1078     ldi w,@1
1079     sts @0,w
1080     .endmacro
1081
1082 .macro INC4          ; increment
1083     ldi w,0xff
1084     sub @3,w
1085     sbc @2,w
1086     sbc @1,w
1087     sbc @0,w
1088     .endmacro
1089 .macro INC3
1090     ldi w,0xff
1091     sub @2,w
1092     sbc @1,w
1093     sbc @0,w
1094     .endmacro
1095 .macro INC2
1096     ldi w,0xff
1097     sub @1,w
1098     sbc @0,w
1099     .endmacro
1100
1101 .macro DEC4          ; decrement
1102     ldi w,0xff
1103     add @3,w
1104     adc @2,w
1105     adc @1,w
1106     adc @0,w
1107     .endmacro
1108 .macro DEC3
1109     ldi w,0xff
1110     add @2,w
1111     adc @1,w
1112     adc @0,w
1113     .endmacro
1114 .macro DEC2
1115     ldi w,0xff

```

```
1116     add @1,w
1117     adc @0,w
1118     .endmacro
1119
1120 .macro CLR9          ; clear (also clears the carry)
1121     sub @0,@0
1122     clr @1
1123     clr @2
1124     clr @3
1125     clr @4
1126     clr @5
1127     clr @6
1128     clr @7
1129     clr @8
1130     .endmacro
1131 .macro CLR8
1132     sub @0,@0
1133     clr @1
1134     clr @2
1135     clr @3
1136     clr @4
1137     clr @5
1138     clr @6
1139     clr @7
1140     .endmacro
1141 .macro CLR7
1142     sub @0,@0
1143     clr @1
1144     clr @2
1145     clr @3
1146     clr @4
1147     clr @5
1148     clr @6
1149     .endmacro
1150 .macro CLR6
1151     sub @0,@0
1152     clr @1
1153     clr @2
1154     clr @3
1155     clr @4
1156     clr @5
1157     .endmacro
1158 .macro CLR5
1159     sub @0,@0
1160     clr @1
1161     clr @2
1162     clr @3
1163     clr @4
1164     .endmacro
1165 .macro CLR4
1166     sub @0,@0
```

```

1167    clr @1
1168    clr @2
1169    clr @3
1170    .endmacro
1171 .macro CLR3
1172     sub @0,@0
1173     clr @1
1174     clr @2
1175     .endmacro
1176 .macro CLR2
1177     sub @0,@0
1178     clr @1
1179     .endmacro
1180
1181 .macro COM4           ; one's complement
1182     com @0
1183     com @1
1184     com @2
1185     com @3
1186     .endmacro
1187 .macro COM3
1188     com @0
1189     com @1
1190     com @2
1191     .endmacro
1192 .macro COM2
1193     com @0
1194     com @1
1195     .endmacro
1196
1197 .macro NEG4           ; negation (two's complement)
1198     com @0
1199     com @1
1200     com @2
1201     com @3
1202     ldi w,0xff
1203     sub @3,w
1204     sbc @2,w
1205     sbc @1,w
1206     sbc @0,w
1207     .endmacro
1208 .macro NEG3
1209     com @0
1210     com @1
1211     com @2
1212     ldi w,0xff
1213     sub @2,w
1214     sbc @1,w
1215     sbc @0,w
1216     .endmacro
1217 .macro NEG2

```

```

1218     com @0
1219     com @1
1220     ldi w,0xff
1221     sub @1,w
1222     sbc @0,w
1223     .endmacro
1224
1225     .macro LDI4      ; r..r0, k ; load immediate
1226         ldi @3, low(@4)
1227         ldi @2, high(@4)
1228         ldi @1,byte3(@4)
1229         ldi @0,byte4(@4)
1230         .endmacro
1231     .macro LDI3
1232         ldi @2, low(@3)
1233         ldi @1, high(@3)
1234         ldi @0,byte3(@3)
1235         .endmacro
1236     .macro LDI2
1237         ldi @1, low(@2)
1238         ldi @0, high(@2)
1239         .endmacro
1240
1241     .macro LDS4          ; load direct from SRAM
1242         lds @3,@4
1243         lds @2,@4+1
1244         lds @1,@4+2
1245         lds @0,@4+3
1246         .endmacro
1247     .macro LDS3
1248         lds @2,@3
1249         lds @1,@3+1
1250         lds @0,@3+2
1251         .endmacro
1252     .macro LDS2
1253         lds @1,@2
1254         lds @0,@2+1
1255         .endmacro
1256
1257     .macro STS4          ; store direct to SRAM
1258         sts @0+0,@4
1259         sts @0+1,@3
1260         sts @0+2,@2
1261         sts @0+3,@1
1262         .endmacro
1263     .macro STS3
1264         sts @0+0,@3
1265         sts @0+1,@2
1266         sts @0+2,@1
1267         .endmacro
1268     .macro STS2

```

```

1269     sts @0+0,@2
1270     sts @0+1,@1
1271     .endmacro
1272
1273 .macro STDZ4 ; d, r3,r2,r1,r0
1274     std z+@0+0,@4
1275     std z+@0+1,@3
1276     std z+@0+2,@2
1277     std z+@0+3,@1
1278     .endmacro
1279 .macro STDZ3 ; d, r2,r1,r0
1280     std z+@0+0,@3
1281     std z+@0+1,@2
1282     std z+@0+2,@1
1283     .endmacro
1284 .macro STDZ2 ; d, r1,r0
1285     std z+@0+0,@2
1286     std z+@0+1,@1
1287     .endmacro
1288
1289 .macro LPM4           ; load program memory
1290     lpm
1291     mov @3,r0
1292     adiw    zl,1
1293     lpm
1294     mov @2,r0
1295     adiw    zl,1
1296     lpm
1297     mov @1,r0
1298     adiw    zl,1
1299     lpm
1300     mov @0,r0
1301     adiw    zl,1
1302     .endmacro
1303 .macro LPM3
1304     lpm
1305     mov @2,r0
1306     adiw    zl,1
1307     lpm
1308     mov @1,r0
1309     adiw    zl,1
1310     lpm
1311     mov @0,r0
1312     adiw    zl,1
1313     .endmacro
1314 .macro LPM2
1315     lpm
1316     mov @1,r0
1317     adiw    zl,1
1318     lpm
1319     mov @0,r0

```

```

1320     adiw    zl,1
1321     .endmacro
1322 .macro LPM1
1323     lpm
1324     mov @0,r0
1325     adiw    zl,1
1326     .endmacro
1327
1328 .macro MOV4          ; move between registers
1329     mov @3,@7
1330     mov @2,@6
1331     mov @1,@5
1332     mov @0,@4
1333     .endmacro
1334 .macro MOV3
1335     mov @2,@5
1336     mov @1,@4
1337     mov @0,@3
1338     .endmacro
1339 .macro MOV2
1340     mov @1,@3
1341     mov @0,@2
1342     .endmacro
1343
1344 .macro ADD4          ; add
1345     add @3,@7
1346     adc @2,@6
1347     adc @1,@5
1348     adc @0,@4
1349     .endmacro
1350 .macro ADD3
1351     add @2,@5
1352     adc @1,@4
1353     adc @0,@3
1354     .endmacro
1355 .macro ADD2
1356     add @1,@3
1357     adc @0,@2
1358     .endmacro
1359
1360 .macro SUB4          ; subtract
1361     sub @3,@7
1362     sbc @2,@6
1363     sbc @1,@5
1364     sbc @0,@4
1365     .endmacro
1366 .macro SUB3
1367     sub @2,@5
1368     sbc @1,@4
1369     sbc @0,@3
1370     .endmacro

```

```

1371 .macro SUB2
1372     sub @1,@3
1373     sbc @0,@2
1374     .endmacro
1375
1376 .macro CP4          ; compare
1377     cp  @3,@7
1378     cpc @2,@6
1379     cpc @1,@5
1380     cpc @0,@4
1381     .endmacro
1382 .macro CP3
1383     cp  @2,@5
1384     cpc @1,@4
1385     cpc @0,@3
1386     .endmacro
1387 .macro CP2
1388     cp  @1,@3
1389     cpc @0,@2
1390     .endmacro
1391
1392 .macro TST4          ; test
1393     clr w
1394     cp  @3,w
1395     cpc @2,w
1396     cpc @1,w
1397     cpc @0,w
1398     .endmacro
1399 .macro TST3
1400     clr w
1401     cp  @2,w
1402     cpc @1,w
1403     cpc @0,w
1404     .endmacro
1405 .macro TST2
1406     clr w
1407     cp  @1,w
1408     cpc @0,w
1409     .endmacro
1410
1411 .macro ADDI4          ; add immediate
1412     subi   @3, low(-@4)
1413     sbci   @2, high(-@4)
1414     sbci   @1,byte3(-@4)
1415     sbci   @0,byte4(-@4)
1416     .endmacro
1417 .macro ADDI3
1418     subi   @2, low(-@3)
1419     sbci   @1, high(-@3)
1420     sbci   @0,byte3(-@3)
1421     .endmacro

```

```

1422 .macro ADDI2
1423     subi    @1,  low(-@2)
1424     sbci    @0,  high(-@2)
1425     .endmacro
1426
1427 .macro SUBI4          ; subtract immediate
1428     subi    @3,  low(@4)
1429     sbci    @2,  high(@4)
1430     sbci    @1,byte3(@4)
1431     sbci    @0,byte4(@4)
1432     .endmacro
1433 .macro SUBI3
1434     subi    @2,  low(@3)
1435     sbci    @1,  high(@3)
1436     sbci    @0,byte3(@3)
1437     .endmacro
1438 .macro SUBI2
1439     subi    @1,  low(@2)
1440     sbci    @0,  high(@2)
1441     .endmacro
1442
1443 .macro LSL5           ; logical shift left
1444     lsl @4
1445     rol @3
1446     rol @2
1447     rol @1
1448     rol @0
1449     .endmacro
1450 .macro LSL4
1451     lsl @3
1452     rol @2
1453     rol @1
1454     rol @0
1455     .endmacro
1456 .macro LSL3
1457     lsl @2
1458     rol @1
1459     rol @0
1460     .endmacro
1461 .macro LSL2
1462     lsl @1
1463     rol @0
1464     .endmacro
1465
1466 .macro LSR4           ; logical shift right
1467     lsr @0
1468     ror @1
1469     ror @2
1470     ror @3
1471     .endmacro
1472 .macro LSR3

```

```

1473     lsr @0
1474     ror @1
1475     ror @2
1476     .endmacro
1477 .macro LSR2
1478     lsr @0
1479     ror @1
1480     .endmacro
1481
1482 .macro ASR4           ; arithmetic shift right
1483     asr @0
1484     ror @1
1485     ror @2
1486     ror @3
1487     .endmacro
1488 .macro ASR3
1489     asr @0
1490     ror @1
1491     ror @2
1492     .endmacro
1493 .macro ASR2
1494     asr @0
1495     ror @1
1496     .endmacro
1497
1498 .macro ROL8           ; rotate left through carry
1499     rol @7
1500     rol @6
1501     rol @5
1502     rol @4
1503     rol @3
1504     rol @2
1505     rol @1
1506     rol @0
1507     .endmacro
1508 .macro ROL7
1509     rol @6
1510     rol @5
1511     rol @4
1512     rol @3
1513     rol @2
1514     rol @1
1515     rol @0
1516     .endmacro
1517 .macro ROL6
1518     rol @5
1519     rol @4
1520     rol @3
1521     rol @2
1522     rol @1
1523     rol @0

```

```
1524     .endmacro
1525 .macro ROL5
1526     rol @4
1527     rol @3
1528     rol @2
1529     rol @1
1530     rol @0
1531     .endmacro
1532 .macro ROL4
1533     rol @3
1534     rol @2
1535     rol @1
1536     rol @0
1537     .endmacro
1538 .macro ROL3
1539     rol @2
1540     rol @1
1541     rol @0
1542     .endmacro
1543 .macro ROL2
1544     rol @1
1545     rol @0
1546     .endmacro
1547
1548 .macro ROR8      ; rotate right through carry
1549     ror @0
1550     ror @1
1551     ror @2
1552     ror @3
1553     ror @4
1554     ror @5
1555     ror @6
1556     ror @7
1557     .endmacro
1558 .macro ROR7
1559     ror @0
1560     ror @1
1561     ror @2
1562     ror @3
1563     ror @4
1564     ror @5
1565     ror @6
1566     .endmacro
1567 .macro ROR6
1568     ror @0
1569     ror @1
1570     ror @2
1571     ror @3
1572     ror @4
1573     ror @5
```

```
1574     .endmacro
1575 .macro R0R5
1576     ror @0
1577     ror @1
1578     ror @2
1579     ror @3
1580     ror @4
1581     .endmacro
1582 .macro R0R4
1583     ror @0
1584     ror @1
1585     ror @2
1586     ror @3
1587     .endmacro
1588 .macro R0R3
1589     ror @0
1590     ror @1
1591     ror @2
1592     .endmacro
1593 .macro R0R2
1594     ror @0
1595     ror @1
1596     .endmacro
1597
1598 .macro PUSH2
1599     push    @0
1600     push    @1
1601     .endmacro
1602 .macro POP2
1603     pop @1
1604     pop @0
1605     .endmacro
1606
1607 .macro PUSH3
1608     push    @0
1609     push    @1
1610     push    @2
1611     .endmacro
1612 .macro POP3
1613     pop @2
1614     pop @1
1615     pop @0
1616     .endmacro
1617
1618 .macro PUSH4
1619     push    @0
1620     push    @1
1621     push    @2
1622     push    @3
1623     .endmacro
1624 .macro POP4
```

```

1625      pop @3
1626      pop @2
1627      pop @1
1628      pop @0
1629      .endmacro
1630
1631 .macro PUSH5
1632     push    @0
1633     push    @1
1634     push    @2
1635     push    @3
1636     push    @4
1637     .endmacro
1638 .macro POP5
1639     pop    @4
1640     pop    @3
1641     pop    @2
1642     pop    @1
1643     pop    @0
1644     .endmacro
1645
1646 ; --- SRAM operations ---
1647 .macro INCS4 ; sram      ; increment SRAM 4-byte variable
1648     lds w,@0
1649     inc w
1650     sts @0,w
1651     brne end
1652     lds w,@0+1
1653     inc w
1654     sts @0+1,w
1655     brne end
1656     lds w,@0+2
1657     inc w
1658     sts @0+2,w
1659     brne end
1660     lds w,@0+3
1661     inc w
1662     sts @0+3,w
1663 end:
1664     .endmacro
1665
1666 .macro INCS3 ; sram      ; increment SRAM 3-byte variable
1667     lds w,@0
1668     inc w
1669     sts @0,w
1670     brne end
1671     lds w,@0+1
1672     inc w
1673     sts @0+1,w
1674     brne end

```

```

1675     lds w,@0+2
1676     inc w
1677     sts @0+2,w
1678 end:
1679     .endmacro
1680
1681 .macro INCS2    ; sram      ; increment SRAM 2-byte variable
1682     lds w,@0
1683     inc w
1684     sts @0,w
1685     brne    end
1686     lds w,@0+1
1687     inc w
1688     sts @0+1,w
1689 end:
1690     .endmacro
1691
1692 .macro INCS     ; sram      ; increment SRAM 1-byte variable
1693     lds w,@0
1694     inc w
1695     sts @0,w
1696     .endmacro
1697
1698 .macro DECS4    ; sram      ; decrement SRAM 4-byte variable
1699     ldi w,1
1700     lds u,@0
1701     sub u,w
1702     sts @0,u
1703     clr w
1704     lds u,@0+1
1705     sbc u,w
1706     sts @0+1,u
1707     lds u,@0+2
1708     sbc u,w
1709     sts @0+2,u
1710     lds u,@0+3
1711     sbc u,w
1712     sts @0+3,u
1713     .endmacro
1714 .macro DECS3    ; sram      ; decrement SRAM 3-byte variable
1715     ldi w,1
1716     lds u,@0
1717     sub u,w
1718     sts @0,u
1719     clr w
1720     lds u,@0+1
1721     sbc u,w
1722     sts @0+1,u
1723     lds u,@0+2
1724     sbc u,w
1725     sts @0+2,u

```

```

1726     .endmacro
1727 .macro DECS2    ; sram      ; decrement SRAM 2-byte variable
1728     ldi w,1
1729     lds u,@0
1730     sub u,w
1731     sts @0,u
1732     clr w
1733     lds u,@0+1
1734     sbc u,w
1735     sts @0+1,u
1736     .endmacro
1737 .macro DECS    ; sram      ; decrement
1738     lds w,@0
1739     dec w
1740     sts @0,w
1741     .endmacro
1742
1743 .macro MOVS4    ; addr0,addr1    ; [addr0] <-- [addr1]
1744     lds w,@1
1745     sts @0,w
1746     lds w,@1+1
1747     sts @0+1,w
1748     lds w,@1+2
1749     sts @0+2,w
1750     lds w,@3+1
1751     sts @0+3,w
1752     .endmacro
1753 .macro MOVS3    ; addr0,addr1    ; [addr0] <-- [addr1]
1754     lds w,@1
1755     sts @0,w
1756     lds w,@1+1
1757     sts @0+1,w
1758     lds w,@1+2
1759     sts @0+2,w
1760     .endmacro
1761 .macro MOVS2    ; addr0,addr1    ; [addr0] <-- [addr1]
1762     lds w,@1
1763     sts @0,w
1764     lds w,@1+1
1765     sts @0+1,w
1766     .endmacro
1767 .macro MOVS    ; addr0,addr1    ; [addr0] <-- [addr1]
1768     lds w,@1
1769     sts @0,w
1770     .endmacro
1771
1772 .macro SEXT    ; reg1,reg0 ; sign extend
1773     clr @0
1774     sbrc   @1,7
1775     dec @0

```

```

1776     .endmacro
1777
1778 ; =====
1779 ;   Jump/Call with constant arguments
1780 ; =====
1781
1782 ; --- calls with arguments a,b,XYZ ---
1783 .macro CX ; subroutine,x
1784     ldi xl, low(@1)
1785     ldi xh,high(@1)
1786     rcall @0
1787     .endmacro
1788 .macro CXY ; subroutine,x,y
1789     ldi xl, low(@1)
1790     ldi xh,high(@1)
1791     ldi yl, low(@2)
1792     ldi yh,high(@2)
1793     rcall @0
1794     .endmacro
1795 .macro CXZ ; subroutine,x,z
1796     ldi xl, low(@1)
1797     ldi xh,high(@1)
1798     ldi zl, low(@2)
1799     ldi zh,high(@2)
1800     rcall @0
1801     .endmacro
1802 .macro CXYZ ; subroutine,x,y,z
1803     ldi xl, low(@1)
1804     ldi xh,high(@1)
1805     ldi yl, low(@2)
1806     ldi yh,high(@2)
1807     ldi zl, low(@3)
1808     ldi zh,high(@3)
1809     rcall @0
1810     .endmacro
1811 .macro CW ; subroutine,w
1812     ldi w, @1
1813     rcall @0
1814     .endmacro
1815 .macro CA ; subroutine,a
1816     ldi a0, @1
1817     rcall @0
1818     .endmacro
1819 .macro CAB ; subroutine,a,b
1820     ldi a0, @1
1821     ldi b0, @2
1822     rcall @0
1823     .endmacro
1824
1825 ; --- jump with arguments w,a,b ---
1826 .macro JW ; subroutine,w

```

```
1827    ldi w, @1
1828    rjmp   @0
1829    .endmacro
1830 .macro JA ; subroutine,a
1831     ldi a0, @1
1832     rjmp   @0
1833     .endmacro
1834 .macro JAB ; subroutine,a,b
1835     ldi a0, @1
1836     ldi b0, @2
1837     rjmp   @0
1838     .endmacro
1839 .list
1840
```

medical_thermo/medical_thermo/lcd.asm

```
1 ; file lcd.asm target ATmega128L-4MHz-STK300
2 ; purpose LCD HD44780U library
3 ; ATmega 128 and Atmel Studio 7.0 compliant
4
5 ; === definitions ===
6 .equ LCD_IR = 0x8000 ; address LCD instruction reg
7 .equ LCD_DR = 0xc000 ; address LCD data register
8
9 ; === subroutines ===
10 LCD_wr_ir:
11 ; in w (byte to write to LCD IR)
12 lds u, LCD_IR ; read IR to check busy flag (bit7)
13 JB1 u,7,LCD_wr_ir ; Jump if Bit=1 (still busy)
14 rcall lcd_4us ; delay to increment DRAM addr counter
15 sts LCD_IR, w ; store w in IR
16 ret
17
18 lcd_4us:
19 rcall lcd_2us ; recursive call
20 lcd_2us:
21 nop ; rcall(3) + nop(1) + ret(4) = 8 cycles (2us)
22 ret
23
24 LCD:
25 LCD_putc:
26 JK a0,CR,LCD_cr ; Jump if a0=CR
27 JK a0,LF,LCD_lf ; Jump if a0=LF
28 LCD_wr_dr:
29 ; in a0 (byte to write to LCD DR)
30 lds w, LCD_IR ; read IR to check busy flag (bit7)
31 JB1 w,7,LCD_wr_dr ; Jump if Bit=1 (still busy)
32 rcall lcd_4us ; delay to increment DRAM addr counter
33 sts LCD_DR, a0 ; store a0 in DR
34 ret
35
36 LCD_clear: JW LCD_wr_ir, 0b00000001 ; clear display
37 LCD_home: JW LCD_wr_ir, 0b00000010 ; return home
38 LCD_cursor_left: JW LCD_wr_ir, 0b00010000 ; move cursor to left
39 LCD_cursor_right: JW LCD_wr_ir, 0b00010100 ; move cursor to right
40 LCD_display_left: JW LCD_wr_ir, 0b00011000 ; shifts display to left
41 LCD_display_right: JW LCD_wr_ir, 0b00011100 ; shifts display to right
42 LCD_blink_on: JW LCD_wr_ir, 0b00001101 ; Display=1,Cursor=0,Blink=1
43 LCD_blink_off: JW LCD_wr_ir, 0b00001100 ; Display=1,Cursor=0,Blink=0
44 LCD_cursor_on: JW LCD_wr_ir, 0b00001110 ; Display=1,Cursor=1,Blink=0
45 LCD_cursor_off: JW LCD_wr_ir, 0b00001100 ; Display=1,Cursor=0,Blink=0
46
47 LCD_init:
48 ; in w,MCUCR ; enable access to ext. SRAM
49 sbr w,(1<<SRE)+(1<<SRW10)
```

```

50    out MCUCR,w
51    CW LCD_wr_ir, 0b00000001 ; clear display
52    CW LCD_wr_ir, 0b00000110 ; entry mode set (Inc=1, Shift=0)
53    CW LCD_wr_ir, 0b00001100 ; Display=1,Cursor=0,Blink=0
54    CW LCD_wr_ir, 0b00111000 ; 8bits=1, 2lines=1, 5x8dots=0
55    ret
56
57 LCD_pos:
58 ; in   a0 = position (0x00..0x0f first line, 0x40..0x4f second line)
59     mov w,a0
60     ori w,0b10000000
61     rjmp   LCD_wr_ir
62
63 LCD_cr:
64 ; moving the cursor to the beginning of the line (carriage return)
65     lds w, LCD_IR          ; read IR to check busy flag (bit7)
66     JB1 w,7,LCD_cr        ; Jump if Bit=1 (still busy)
67     andi  w,0b01000000      ; keep bit6 (begin of line 1/2)
68     ori w,0b10000000      ; write address command
69     rcall  lcd_4us         ; delay to increment DRAM addr counter
70     sts LCD_IR,w          ; store in IR
71     ret
72
73 LCD_lf:
74 ; moving the cursor to the beginning of the line 2 (line feed)
75     push   a0              ; safeguard a0
76     ldi a0,$40             ; load position $40 (begin of line 2)
77     rcall  LCD_pos         ; set cursor position
78     pop    a0              ; restore a0
79     ret

```

medical_thermo/medical_thermo/printf.asm

```
1 ; file printf.asm target ATmega128L-4MHz-STK300
2 ; purpose library, formatted output generation
3 ; v2019.02 20180821 supports SRAM input from 0x0260
4 ;           through 0x02ff that should be reserved
5 ;
6 ; === description ===
7 ;
8 ; The program "printf" interprets and prints formatted strings.
9 ; The special formatting characters recognized are:
10 ;
11 ; FDEC decimal number
12 ; FHEX hexadecimal number
13 ; FBIN binary number
14 ; FFRAC fixed fraction number
15 ; FCHAR single ASCII character
16 ; FSTR zero-terminated ASCII string
17 ;
18 ; The special formatting characters are distinguished from normal
19 ; ASCII characters by having their bit7 set to 1.
20 ;
21 ; Signification of bit fields:
22 ;
23 ; b    bytes      1..4 b bytes      2
24 ; s    sign        0(unsigned), 1(signed) 1
25 ; i    integer digits
26 ; e    base        2,,36          5
27 ; dp   dec. point 0..32          5
28 ; $if  i=integer digits, 0=all digits, 1..15 digits
29 ;       f=fraction digits, 0=no fraction, 1..15 digits
30 ;
31 ; Formatting characters must be followed by an SRAM address (0..ff)
32 ; that determines the origin of variables that must be printed (if any)
33 ; FBIN, sram
34 ; FHEX, sram
35 ; FDEC, sram
36 ; FCHAR,sram
37 ; FSTR, sram
38 ;
39 ; The address 'sram' is a 1-byte constant. It addresses
40 ;     0..1f registers r0..r31,
41 ;     20..3f i/o ports, (need to be addressed with an offset of $20)
42 ;     0x0260..0x02ff SRAM
43 ; Variables can be located into register and I/Os, and can also
44 ; be stored into data SRAM at locations 0x0200 through 0x02ff. Any
45 ; sram address higher than 0x0060 is assumed to be at (0x0260+address)
46 ; from automatic address detection in _printf_formatted: and subsequent
47 ; assignment to xh; xl keeps its value. Consequently, variables that are
48 ; to be stored into SRAM and further printed by fprint must reside at
49 ; 0x0200 up to 0x02ff, and must be addressed using a label. Usage: see
```

```

50 ; file string1.asm, for example.
51
52 ; The FFRAC formatting character must be followed by
53 ;   ONE sram address and
54 ;   TWO more formatting characters
55 ; FFRAC,sram,dp,$if
56
57 ; dp    decimal point position, 0=right, 32=left
58 ; $if   format i.f, i=integer digits, f=fraction digits
59
60 ; The special formatting characters use the following coding
61 ;
62 ; FDEC 11bb'iiis   i=0 all digits, i=1-7 digits
63 ; FBIN 101i'iiis   i=0 8 digits,   i=1-7 digits
64 ; FHEX 1001'iiis   i=0 8 digits,   i=1-7 digits
65 ; FFRAC 1000'1bbs
66 ; FCHAR 1000'0100
67 ; FSTR  1000'0101
68 ; FREP  1000'0110
69 ; FFUNC 1000'0111
70 ;   1000'0010
71 ;   1000'0011
72 ; FESC  1000'0000
73
74 ; examples
75 ; formatting string      printing
76 ; "a=",FDEC,a,0          1-byte variable a, unsigned decimal
77 ; "a=",FDEC2,a,0         2-byte variable a (a1,a0), unsigend
78 ; "a=",FDEC|FSIGN,a,0    1-byte variable 1, signed decimal
79 ; "n=",FBIN,PIND+$20,0   i/o port, binary, notice offset of $20
80 ; "f=",FFRAC4|FSIGN,a,16,$88,0 4-byte signed fixed-point fraction
81 ;                      dec.point at 16, 8 int.digits, 8 frac.digits
82 ; "f=",FFRAC2,a,16,$18,0   2-byte unsigned fixed-point fraction
83 ;                      dec.point at 16, 1 int.digits, 8 frac.digits
84 ; "a=",FDEC|FDIG5|FSIGN,a,0 1-byte variable, 5-digit, decimal, signed
85 ; "a=",FDEC|FDIG5,a,0      1-byte variable, 5-digit, decimal, unsigned
86
87 ; === registers modified ===
88 ; e0,e1 used to transmit address of putc routine
89 ; zh,zl used as pointer to prog-memory
90
91 ; === constants =====
92
93 .equ FDEC    = 0b11000000 ; 1-byte variable
94 .equ FDEC2   = 0b11010000 ; 2-byte variable
95 .equ FDEC3   = 0b11100000 ; 3-byte variable
96 .equ FDEC4   = 0b11110000 ; 4-byte variable
97
98 .equ FBIN    = 0b10100000
99 .equ FHEX    = 0b10010100 ; 1-byte variable
100 .equ FHEX2   = 0b10011000 ; 2-byte variable

```

```

101 .equ FHEX3 = 0b10011100 ; 3-byte variable
102 .equ FHEX4 = 0b10010000 ; 4-byte variable
103
104 .equ FFRAC = 0b10001000 ; 1-byte variable
105 .equ FFRAC2 = 0b10001010 ; 2-byte variable
106 .equ FFRAC3 = 0b10001100 ; 3-byte variable
107 .equ FFRAC4 = 0b10001110 ; 4-byte variable
108
109 .equ FCHAR = 0b10000100
110 .equ FSTR = 0b10000101
111
112 .equ FSIGN = 0b00000001
113
114 .equ FDIG1 = 1<<1
115 .equ FDIG2 = 2<<1
116 .equ FDIG3 = 3<<1
117 .equ FDIG4 = 4<<1
118 .equ FDIG5 = 5<<1
119 .equ FDIG6 = 6<<1
120 .equ FDIG7 = 7<<1
121
122 ; ===macro =====
123
124 .macro PRINTF ; putc function (UART, LCD...)
125     ldi w, low(@0) ; address of "putc" in e1:d0
126     mov e0,w
127     ldi w,high(@0)
128     mov e1,w
129     rcall _printf
130     .endmacro
131
132 ; mod y,z
133
134
135 ; === routines =====
136
137 _printf:
138     POPZ ; z points to begin of "string"
139     MUL2Z ; multiply Z by two, (word ptr -> byte ptr)
140     PUSHX
141
142 _printf_read:
143     lpm ; places prog_mem(Z) into r0 (=c)
144     adiw zl,1 ; increment pointer Z
145     tst r0 ; test for ZERO (=end of string)
146     breq _printf_end ; char=0 indicates end of ascii string
147     brmi _printf_formatted ; bit7=1 indicates formatting character
148     mov w,r0
149     rcall _putw ; display the character
150     rjmp _printf_read ; read next character in the string
151

```

```

152 _printf_end:
153     adiw    zl,1      ; point to the next character
154     DIV2Z          ; divide by 2 (byte ptr -> word ptr)
155     POPX
156     ijmp          ; return to instruction after "string"
157
158 _printf_formatted:
159
160 ; FDEC 11bb'iiis
161 ; FBIN 101i'iiis
162 ; FHEX 1001'iiis
163 ; FFRAC 1000'1bbs
164 ; FCHAR 1000'0100
165 ; FSTR 1000'0101
166
167     bst r0,0        ; store sign in T
168     mov w,r0        ; store formatting character in w
169     lpm
170     mov xl,r0        ; load x-pointer with SRAM address
171     cpi xl,0x60
172     brlo rio_space
173 dataram_space:       ; variable originates from SRAM memory
174     ldi xh,0x02      ;>addresses are limited to 0x0260 through 0x02ff
175     rjmp space_detect_end  ;>that enables automatic detection of the origin
176 rio_space:          ; variable originates from reg or I/O space
177     clr xh          ; clear high-byte, addresses are 0x0000 through 0x003f
(0x005f)
178 space_detect_end:
179     adiw    zl,1      ; increment pointer Z
180
181 ;   JB1 w,6,_putdec
182 ;   JB1 w,5,_putbin
183 ;   JB1 w,4,_puthex
184 ;   JB1 w,3,_putfrac
185     JK  w,FCHAR,_putchar
186     JK  w,FSTR ,_putstr
187     rjmp   _putnum
188
189     rjmp   _printf_read
190
191 ; === putc (put character) =====
192 ; in   w   character to put
193 ; e1,e0   address of output routine (UART, LCD putc)
194 _putw:
195     PUSH3  a0,zh,zl
196     MOV3   a0,zh,zl, w,e1,e0
197     icall          ; indirect call to "putc"
198     POP3   a0,zh,zl
199     ret
200
201 ; === putchar (put character) =====

```

```

202 ; in    x  pointer to character to put
203 _putchar:
204     ld    w,x
205     rcall _putw
206     rjmp  _printf_read
207
208 ; === putstr (put string) =====
209 ; in    x  pointer to ascii string
210 ;   b3,b2  address of output routine (UART, LCD putc)
211 _putstr:
212     ld    w,x+
213     tst   w
214     brne PC+2
215     rjmp  _printf_read
216     rcall _putw
217     rjmp  _putstr
218
219 ; === putnum (dec/bin/hex/frac) =====
220 ; in    x  pointer to SRAM variable to print
221 ;   r0  formatting character
222
223 _putnum:
224     PUSH4  a3,a2,a1,a0 ; safeguard a
225     PUSH4  b3,b2,b1,b0 ; safeguard b
226     LDX4   a3,a2,a1,a0 ; load operand to print into a
227
228 ; FDEC  11bb'iiis
229 ; FBIN   101i'iiis
230 ; FHEX   1001'iiis
231 ; FRACT  1000'1bbs
232
233     JB1  w,6,_putdec
234     JB1  w,5,_putbin
235     JB1  w,4,_puthex
236     JB1  w,3,_putfrac
237
238 ; FDEC  11bb'iiis
239 _putdec:
240     ldi b0,10      ; b0 = base (10)
241
242     mov b1,w
243     lsr b1
244     andi b1,0b111
245     swap  b1      ; b1 = format 0iii'0000 (integer digits)
246     ldi b2,0      ; b2 = dec. point position = 0 (right)
247
248     mov b3,w
249     swap  b3
250     andi b3,0b11
251     inc b3       ; b3 = number of bytes (1..4)
252     rjmp  _getnum ; get number of digits (iii)

```

```

253
254 ; FBIN 101i'iiis    addr
255 _putbin:
256     ldi b0,2          ; b0 = base (2)
257     ldi b3,4          ; b3 = number of bytes (4)
258     rjmp   _getdig ; get number of digits (iii)
259
260 ; FHEX 1001'iiis    addr
261 _puthex:
262     ldi b0,16         ; b0 = base (16)
263     ldi b3,4          ; b3 = number of bytes (4)
264     rjmp   _getdig
265
266 _getdig:
267     mov b1,w
268     lsr b1
269     andi  b1,0b111
270     brne  PC+2
271     ldi b1,8          ; if b1=0 then 8-digits
272     swap   b1          ; b1 = format 0iii'0000 (integer digits)
273     ldi b2, 0           ; b2 = dec. point position = 0 (right)
274     rjmp   _getnum
275
276 ; FFRAC 1000'1bbs    addr      00dd'dddd,      iii'i'ffff
277
278 _putfrac:
279     ldi b0,10          ; base=10
280     lpm
281     mov b2,r0          ; load dec.point position
282     adiw   zl,1          ; increment char pointer
283     lpm
284     mov b1,r0          ; load ii.ff format
285     adiw   zl,1          ; increment char pointer
286
287     mov b3,w
288     asr b3
289     andi  b3,0b11
290     inc b3            ; b3 = number of bytes (1..4)
291
292     rjmp   _getnum
293
294 _getnum:
295 ; in   a  4-byte variable
296 ;   b3  number of bytes (1..4)
297 ;   T   sign, 0=unsigned, 1=signed
298
299     JK  b3,4,_printf_4b
300     JK  b3,3,_printf_3b
301     JK  b3,2,_printf_2b
302
303 _printf_1b:        ; sign extension

```

```

304     clr a1
305     brtc  PC+3    ; T=1 sign extension
306     sbrc  a0,7
307     ldi a1,0xff
308 _printf_2b:
309     clr a2
310     brtc  PC+3    ; T=1 sign extension
311     sbrc  a1,7
312     ldi a2,0xff
313 _printf_3b:
314     clr a3
315     brtc  PC+3    ; T=1 sign extension
316     sbrc  a2,7
317     ldi a3,0xff
318 _printf_4b:
319
320     rcall  _ftoa      ; float to ascii
321     POP4   b3,b2,b1,b0 ; restore b
322     POP4   a3,a2,a1,a0 ; restore a
323
324     rjmp   _printf_read
325
326 ; =====
327 ; func ftoa
328 ; converts a fixed-point fractional number to an ascii string
329 ;
330 ; in   a3-a0  variable to print
331 ; b0  base, 2 to 36, but usually decimal (10)
332 ; b1  number of digits to print ii.ff
333 ; b2  position of the decimal point (0=right, 32=left)
334 ; T   sign (T=0 unsigned, T=1 signed)
335
336 _ftoa:
337     push   d0
338     PUSH4  c3,c2,c1,c0 ; c = fraction part, a = integer part
339     CLR4   c3,c2,c1,c0 ; clear fraction part
340
341     brtc   _ftoa_plus ; if T=0 then unsigned
342     clt
343     tst a3            ; if MSb(a)=1 then a=-a
344     brpl   _ftoa_plus
345     set                ; T=1 (minus)
346     tst b1
347     breq   PC+2        ; if b1=0 then print ALL digits
348     subi   b1,0x10       ; decrease int digits
349     NEG4   a3,a2,a1,a0 ; negate a
350 _ftoa_plus:
351     tst b2            ; b0=0 (only integer part)
352     breq   _ftoa_int
353 _ftoa_shift:
354     ASR4   a3,a2,a1,a0 ; a = integer part

```

```

355     ROR4    c3,c2,c1,c0 ; c = fraction part
356     DJNZ    b2,_ftoa_shift
357 _ftoa_int:
358     push    b1           ; ii.ff (ii=int digits)
359     swap    b1
360     andi    b1,0x0f
361
362     ldi w,'.'          ; push decimal point
363     push    w
364 _ftoa_int1:
365     rcall   _div41      ; int=int/10
366     mov w,d0            ; d=reminder
367     rcall   _hex2asc
368     push    w           ; push rem(int/10)
369     TST4    a3,a2,a1,a0 ; (int/10)=?
370     breq    _ftoa_space ; (int/10)=0 then finished
371     tst b1
372     breq    _ftoa_int1 ; if b1=0 then print ALL int-digits
373     DJNZ    b1,_ftoa_int1
374     rjmp    _ftoa_sign
375 _ftoa_space:
376     tst b1              ; if b1=0 then print ALL int-digits
377     breq    _ftoa_sign
378     dec b1
379     breq    _ftoa_sign
380     ldi w,' '           ; write spaces
381     rcall   _putw
382     rjmp    _ftoa_space
383 _ftoa_sign:
384     brtc   PC+3          ; if T=1 then write 'minus'
385     ldi w,'-'
386     rcall   _putw
387 _ftoa_int3:
388     pop w
389     cpi w,'.'
390     breq    PC+3
391     rcall   _putw
392     rjmp    _ftoa_int3
393
394     pop b1              ; ii.ff (ff=frac digits)
395     andi    b1,0x0f
396     tst b1
397     breq    _ftoa_end
398 _ftoa_point:
399     rcall   _putw          ; write decimal point
400     MOV4    a3,a2,a1,a0, c3,c2,c1,c0
401 _ftoa_frac:
402     rcall   _mul41        ; d.frac=10*frac
403     mov w,d0
404     rcall   _hex2asc
405     rcall   _putw

```

```

406    DJNZ    b1,_ftoa_frac
407 _ftoa_end:
408     POP4    c3,c2,c1,c0
409     pop d0
410     ret
411
412 ; === hexadecimal to ascii ===
413 ; in   w
414 _hex2asc:
415     cpi w,10
416     brsh   PC+3
417     addi   w, '0'
418     ret
419     addi   w, ('a'-10)
420     ret
421
422 ; === multiply 4byte*1byte ===
423 ; funct mul41
424 ; multiplies a3-a0 (4-byte) by b0 (1-byte)
425 ;
426 ; in   a3..a0 multiplicand (argument to multiply)
427 ; b0   multiplier
428 ; out  a3..a0 result
429 ; d0   result MSB (byte 4)
430 ;
431 _mul41: clr d0           ; clear byte4 of result
432     ldi w,32             ; load bit counter
433 __m41:  clc              ; clear carry
434     sbrc   a0,0           ; skip addition if LSB=0
435     add d0,b0             ; add b to MSB of a
436     ROR5    d0,a3,a2,a1,a0 ; shift-right c, LSB (of b) into carry
437     DJNZ    w,__m41        ; Decrement and Jump if bit-count Not Zero
438     ret
439
440 ; === divide 4byte/1byte ===
441 ; func div41
442 ; in   a0..a3 divident (argument to divide)
443 ; b0   divider
444 ; out  a0..a3 result
445 ; d0   remainder
446 ;
447 _div41: clr d0           ; d will contain the remainder
448     ldi w,32             ; load bit counter
449 __d41:  ROL5    d0,a3,a2,a1,a0 ; shift carry into result c
450     sub d0, b0             ; subtract b from remainder
451     brcc   PC+2
452     add d0, b0             ; restore if remainder became negative
453     DJNZ    w,__d41        ; Decrement and Jump if bit-count Not Zero
454     ROL4    a3,a2,a1,a0 ; last shift (carry into result c)
455     COM4    a3,a2,a1,a0 ; complement result
456     ret

```

457 |

medical_thermo/medical_thermo/encoder.asm

```
1 ; file encoder.asm target ATmega128L-4MHz-STK300
2 ; purpose library angular encoder operation
3
4 ; === definitions ===
5 .equ ENCOD = PORTE
6
7 .dseg
8 enc_old:.byte 1
9 .cseg
10
11 ; === routines ===
12
13 encoder_init:
14     in w,ENCOD-1      ; make 3 lines input
15     andi w,0b10001111
16     out ENCOD-1,w
17     in w,ENCOD        ; enable 3 internal pull-ups
18     ori w,0b01110000
19     out ENCOD,w
20     ret
21
22 encoder:
23 ; a0,b0 if button=up then increment/decrement a0
24 ; a0,b0 if button=down then incremnt/decrement b0
25 ; T    T=1 button press (transition up-down)
26 ; Z Z=1 button down change
27
28     clt                  ; preclear T
29     in _w,ENCOD-2       ; read encoder port (_w=new)
30
31     andi _w,0b01110000  ; mask encoder lines (A,B,I)
32     lds _u,enc_old       ; load prevous value (_u=old)
33     cp _w,_u             ; compare new>old ?
34     brne PC+3
35     clz
36     ret                  ; if new=old then return (Z=0)
37     sts enc_old,_w       ; store encoder value for next time
38
39     eor _u,_w            ; exclusive or detects transitions
40     clz                  ; clear Z flag
41     sbrc _u,ENCOD_I
42     rjmp encoder_button ; transition on I (button)
43     sbrs _u,ENCOD_A
44     ret                  ; return (no transition on I or A)
45
46     sbrs _w,ENCOD_I      ; is the button up or down ?
47     rjmp i_down
48 i_up:
49     sbrc _w,ENCOD_A
```

```

50      rjmp    a_rise
51 a_fall:
52     inc a0          ; if B=1 then increment
53     sbrs _w,ENCOD_B
54     subi a0,2       ; if B=0 then decrement
55     rjmp    i_up_done
56 a_rise:
57     inc a0          ; if B=0 then increment
58     sbrc _w,ENCOD_B
59     subi a0,2       ; if B=1 then decrement
60 i_up_done:
61     clz              ; clear Z
62     ret
63
64 i_down:
65     sbrc _w,ENCOD_A
66     rjmp    a_rise2
67 a_fall2:
68     inc b0          ; if B=1 then increment
69     sbrs _w,ENCOD_B
70     subi b0,2       ; if B=0 then decrement
71     rjmp    i_down_done
72 a_rise2:
73     inc b0          ; if B=0 then increment
74     sbrc _w,ENCOD_B
75     subi b0,2       ; if B=1 then decrement
76 i_down_done:
77     sez              ; set Z
78     ret
79
80 encoder_update:
81     ; save old a0
82     mov r16, a0
83
84     ; call the existing routine
85     rcall encoder
86
87     ; now a0 = new count, r16 = old count
88     mov r17, a0
89     ; compute ? = new ? old
90     sub r17, r16      ; r17 = a0_new ? a0_old
91     mov r15, r17      ; put result in r15, us
92
93     ret
94
95 encoder_button:
96     sbrc _w,ENCOD_I
97     rjmp    i_rise
98 i_fall:
99     set               ; set T=1 to indicate button press
100    ret

```

```
101 i_rise:  
102     ret  
103  
104 .macro CYCLIC ;reg,lo,hi  
105     cpi @0,@1-1  
106     brne    PC+2  
107     ldi @0,@2  
108     cpi @0,@2+1  
109     brne    PC+2  
110     ldi @0,@1  
111 .endmacro  
112
```

medical_thermo/medical_thermo/wire1.asm

```
1 ; file wire1.asm    target ATmega128L-4MHz-STK300
2 ; purpose Dallas 1-wire(R) interface library
3
4 ; === definitions ===
5 .equ DQ_port = PORTB
6 .equ DQ_pin = DQ
7
8 .equ DS18B20      = 0x28
9
10 .equ readROM     = 0x33
11 .equ matchROM    = 0x55
12 .equ skipROM     = 0xcc
13 .equ searchROM   = 0xf0
14 .equ alarmSearch = 0xec
15
16 .equ writeScratchpad = 0x4e
17 .equ readScratchpad = 0xbe
18 .equ copyScratchpad = 0x48
19 .equ convertT     = 0x44
20 .equ recallE2     = 0xb8
21 .equ readPowerSupply = 0xb4
22
23 ; === routines ===
24
25 .macro WIRE1 ; t0,t1,t2
26     cli                      ; !! interrupts off
27     sbi DQ_port-1,DQ_pin    ; pull DQ low (DDR=1 output)
28     ldi w,(@0+1)/2
29     rcall wire1_wait        ; wait low time (t0)
30     cbi DQ_port-1,DQ_pin    ; release DQ (DDR=0 input)
31     ldi w,(@1+1)/2
32     rcall wire1_wait        ; wait high time (t1)
33     in  w,DQ_port-2         ; sample line (PINx=PORTx-2)
34     bst w,DQ_pin            ; store result in T
35     ldi w,(@2+1)/2
36     rcall wire1_wait        ; wait separation time (t2)
37     sei                      ; interrupts back on
38     ret
39     .endmacro
40
41 wire1_wait:
42     dec w                  ; loop time 2usec
43     nop
44     nop
45     nop
46     nop
47     nop
48     brne wire1_wait
49     ret
```

```

50
51 wire1_init:
52     cbi DQ_port, DQ_pin      ; PORT=0 low (for pull-down)
53     cbi DQ_port-1,DQ_pin    ; DDR=0 (input hi Z)
54     ret
55
56 wire1_reset:    WIRE1    480,70,410
57 wire1_write0:   WIRE1    56,4,1
58 wire1_write1:   WIRE1    1,59,1
59 wire1_read1:   WIRE1    1,14,45
60
61 wire1_write:
62     push    a1
63     ldi a1,8
64     ror a0
65
66     brcc    PC+3           ; if C=1 then wire1, else wire0
67     rcall   wire1_write1
68     rjmp    PC+2
69     rcall   wire1_write0
70
71     DJNZ    a1,wire1_write+2 ; dec and jump if not zero
72     pop    a1
73     ret
74
75 wire1_read:
76     push    a1
77     ldi a1,8
78     ror a0
79     rcall   wire1_read1      ; returns result in T
80     bld a0,7                 ; move T to MSb
81     DJNZ    a1,wire1_read+2 ; dec and jump if not zero
82     pop    a1
83     ret
84
85 wire1_crc:
86     ldi w,0b00011001
87     ldi a2,8
88 crc1:    ror a0
89     brcc    PC+2
90     eor a1,w
91     bst a1,0
92     ror a1
93     bld a1,7
94     DJNZ    a2,crc1
95     ret
96
97

```

medical_thermo/medical_thermo/ws2812_driver.asm

```
1 ;=====
2 ; WS2812B RGB LED MATRIX DRIVER
3 ;=====
4 ; Target: ATmega128L @ 4MHz
5 ;
6 ; Description:
7 ; This driver provides bit-banging control for WS2812B addressable RGB
8 ; LEDs arranged in an 8x8 matrix configuration. It handles precise timing
9 ; requirements and provides helper functions for matrix addressing.
10 ;
11 ; WS2812B Protocol Specifications:
12 ; - Single-wire interface using non-return-to-zero (NRZ) coding
13 ; - Strict timing: "0" bit = 0.4μs high + 0.85μs low
14 ;           "1" bit = 0.8μs high + 0.45μs low
15 ; - Reset condition: >50μs low
16 ; - Data sent in GRB order (not RGB)
17 ; - 24 bits per LED (8 bits per color channel)
18 ;
19 ; Functions:
20 ; - ws_init: Initialize WS2812 pin as output
21 ; - ws_byte3wr: Transmit one RGB pixel (3 bytes)
22 ; - ws_reset: Latch data to LEDs
23 ; - ws_idx_xy: Convert X,Y coordinates to linear index
24 ; - ws_offset_idx: Calculate buffer address from linear index
25 ;
26 ; Last Modified: May 25, 2025
27 ;=====

28
29 ; file ws2812_driver.asm      target ATmega128L-4 MHz
30 ; purpose: reusable bit-bang driver + 8x8 XY helpers for WS2812B
31 ;
32 .equ WS_PORT_REG = PORTD
33 .equ WS_DDR_REG = DDRD
34 .equ WS_PIN_IDX = 7
35 .equ WS_PIN_MASK = (1 << WS_PIN_IDX)

36
37 .equ WS_BUF_BASE = 0x0400      ; 8x8x3-byte frame buffer

38
39 ;-----
40 ; REGISTER PRESERVATION – Save/restore registers during function calls
41 ;-----

42     .macro WS_PUSH_ALL
43         push    a0
44         push    a1
45         push    a2
46         push    u
47         push    w
48         push    r24
49         push    r25
```

```

50          push    ZH
51          push    ZL
52      .endm
53
54      .macro WS_POP_ALL
55          pop     ZL
56          pop     ZH
57          pop     r25
58          pop     r24
59          pop     w
60          pop     u
61          pop     a2
62          pop     a1
63          pop     a0
64      .endm
65
66 ;-----
67 ;  BIT TRANSMISSION MACROS – Precisely timed bit patterns for WS2812B
68 ;-----
69 ; "0" bit total ≈ 5 cycles (T0H ≈ 0.40 µs, T0L ≈ 0.85 µs)
70 .macro WS_WR0
71     clr    u                      ; u = 0 (destroys u **inside** routine)
72     sbi   WS_PORT_REG, WS_PIN_IDX ; high      (2 cy)
73     out   WS_PORT_REG, u         ; low       (1 cy) full-port write
74     nop                           ; 1 cy
75     nop                           ; 1 cy
76 .endm
77 ; "1" bit total ≈ 8 cycles (T1H ≈ 0.80 µs, T1L ≈ 0.45 µs)
78 .macro WS_WR1
79     sbi   WS_PORT_REG, WS_PIN_IDX ; high      (2 cy)
80     nop                           ; 1 cy
81     nop                           ; 1 cy
82     cbi   WS_PORT_REG, WS_PIN_IDX ; low       (2 cy)
83 .endm
84
85 ;=====
86 ;  PUBLIC INTERFACE FUNCTIONS
87 ;=====
88
89 ;-----
90 ;  INITIALIZATION – Configure data pin for WS2812 control
91 ;-----
92 ; set the data pin as output
93 ws_init:
94     OUTI WS_DDR_REG, WS_PIN_MASK
95     ret
96
97 ;-----
98 ;  DATA TRANSMISSION – Send color data to WS2812 LEDs
99 ;-----
100 ; bit-bang three bytes (G=a0, R=a1, B=a2)

```

```

101 ; u & w pushed so the caller never sees them modified
102 ws_byte3wr:
103     push  u
104     push  w
105
106     ; - byte G (a0) -
107     ldi   w, 8
108 _b0:
109     sbrc  a0, 7
110     rjmp  _b0_1
111     WS_WR0
112     rjmp  _b0_next
113 _b0_1:
114     WS_WR1
115 _b0_next:
116     lsl   a0
117     dec   w
118     brne  _b0
119
120     ; - byte R (a1) -
121     ldi   w, 8
122 _b1:
123     sbrc  a1, 7
124     rjmp  _b1_1
125     WS_WR0
126     rjmp  _b1_next
127 _b1_1:
128     WS_WR1
129 _b1_next:
130     lsl   a1
131     dec   w
132     brne  _b1
133
134     ; - byte B (a2) -
135     ldi   w, 8
136 _b2:
137     sbrc  a2, 7
138     rjmp  _b2_1
139     WS_WR0
140     rjmp  _b2_next
141 _b2_1:
142     WS_WR1
143 _b2_next:
144     lsl   a2
145     dec   w
146     brne  _b2
147
148     pop   w
149     pop   u
150     ret
151

```

```

152 ;-----
153 ; FRAME LATCHING - Signal end of frame to update LED display
154 ;-----
155 ; hold the data line low ≥50 µs to latch the frame
156 ws_reset:
157     cbi WS_PORT_REG, WS_PIN_IDX
158     WAIT_US 50
159     ret
160
161 ;-----
162 ; COORDINATE CONVERSION - Matrix addressing utilities
163 ;-----
164 ; r24=x, r25=y → r24 = x + 8xy    (clobbers u, r24)
165 ws_idx_xy:
166     mov u, r25
167     lsl u
168     lsl u
169     lsl u
170     add r24, u
171     ret
172
173 ; r24=index → Z = WS_BUF_BASE + 3×index  (clobbers u, w, ZL, ZH)
174 ws_offset_idx:
175     mov w, r24          ; w = idx
176     lsl w              ; w = 2×idx
177     mov u, r24
178     add w, u          ; w = 3×idx
179     ldi ZL, low(WS_BUF_BASE)
180     ldi ZH, high(WS_BUF_BASE)
181     add ZL, w
182     clr u
183     adc ZH, u          ; add carry
184     ret

```

medical_thermo/medical_thermo/ws2812_helpers.asm

```
1 ;=====
2 ; WS2812B RGB LED MATRIX HELPER FUNCTIONS
3 ;=====
4 ; Target: ATmega128L @ 4MHz
5 ;
6 ; Description:
7 ; This file provides higher-level helper functions for working with
8 ; the WS2812B 8x8 RGB LED matrix. It implements common operations like
9 ; filling the entire matrix with a single color, leveraging the lower-level
10 ; driver functions from ws2812_driver.asm.
11 ;
12 ; Dependencies:
13 ; - ws2812_driver.asm must be included before this file
14 ; - Requires the WS_BUF_BASE memory area defined in ws2812_driver.asm
15 ; - Uses WAIT_US macro from macros.asm
16 ;
17 ; Functions:
18 ; - matrix_solid: Fill entire 8x8 matrix with a single RGB color
19 ;
20 ; Register Usage:
21 ; - Input: a0=Green, a1=Red, a2=Blue (GRB order to match WS2812B protocol)
22 ; - Preserves: r22, ZL, ZH
23 ; - Clobbers: Z, r0, w=r16 (all scratch registers per calling convention)
24 ;
25 ; Last Modified: May 25, 2025
26 ;=====

27 matrix_solid:
28     push    r22
29     push    ZL
30     push    ZH
31
32 ;-----
33 ; BUFFER PREPARATION – Fill frame buffer with specified color
34 ;-----
35     ldi    ZL, low(WS_BUF_BASE)
36     ldi    ZH, high(WS_BUF_BASE)
37     ldi    r22, 64          ; 64 pixels (8x8 matrix)
38 m_fill_loop:
39     st     Z+, a0          ; Store Green component
40     st     Z+, a1          ; Store Red component
41     st     Z+, a2          ; Store Blue component
42     dec   r22              ; Decrement pixel counter
43     brne  m_fill_loop     ; Continue until all pixels filled
44
45 ;-----
46 ; DATA TRANSMISSION – Send buffer contents to LED matrix
47 ;-----
48     ldi    ZL, low(WS_BUF_BASE)      ; Reset Z pointer to buffer start
49     ldi    ZH, high(WS_BUF_BASE)
```

```

50      _LDI    r0, 64          ; 64 pixels to transmit
51  m_send_loop:
52      ld     a0, Z+          ; Load Green component
53      ld     a1, Z+          ; Load Red component
54      ld     a2, Z+          ; Load Blue component
55      cli                  ; Disable interrupts for precise
      timing
56      rcall   ws_byte3wr    ; Transmit one RGB pixel
57      sei                  ; Re-enable interrupts
58      dec    r0              ; Decrement pixel counter
59      brne   m_send_loop    ; Continue until all pixels sent
60      rcall   ws_reset      ; Send reset pulse to latch data
61
62  ;-----
63  ;  CLEANUP – Restore preserved registers and return
64  ;-----
65      pop    ZH              ; Restore Z pointer
66      pop    ZL              ; Restore counter register
67      pop    r22             ; Return to caller
68

```

medical_thermo/medical_thermo/main.asm

```
1 ;=====
2 ; MEDICAL CONSOLE - Main Control Program
3 ;=====
4 ; Target Hardware: ATmega128L @ 4 MHz on STK-300 Development Board
5 ;
6 ; Program Description:
7 ; - Multi-state medical console with temperature monitoring
8 ; - Implements a finite state machine with multiple operating modes
9 ; - Includes games, diagnostic tools, and temperature monitoring
10 ; - Uses DS18B20 digital temperature sensor via 1-Wire protocol
11 ; - Features WS2812 RGB LED matrix for visual feedback
12 ; - User input via rotary encoder and push buttons
13 ;
14 ; Last Modified: May 25, 2025
15 ;=====

16 ;
17 ;-----
18 ; INCLUDES - Core system definitions and macros
19 ;-----
20         .include "m128def.inc"      ; ATmega128 register definitions
21         .include "definitions.asm" ; Global constants and registers
22         .include "macros.asm"     ; Utility macros for code simplification
23
24 ;-----
25 ; GLOBAL REGISTERS AND CONSTANTS
26 ;-----
27         .def   sel = r6           ; Current FSM state register
28         .def   s   = r14          ; Stable scratch register for operations
29
30         ; System State Constants
31         .equ   FLG_TEMP  = 0      ; Bit0 of 'flags' - temperature-ready flag
32         .equ   REG_STATES = 4      ; Number of valid game states (0-3)
33         .equ   ST_HOME   = 0      ; Home/idle state
34         .equ   ST_GAME1  = 1      ; Snake game state
35         .equ   ST_GAME2  = 2      ; Game 2 state
36         .equ   ST_GAME3  = 3      ; Game 3 state
37         .equ   ST_DOCTOR = 4      ; Diagnostic/doctor mode
38
39         ; Timer and Hardware Constants
40         .equ   T1_PREH  = 0xF0    ; Timer-1 preload high byte
41         .equ   T1_PREL  = 0xBE    ; Timer-1 preload low byte (approx 1s)
42         .equ   LED_BIT   = 7      ; PF7 heartbeat indicator (active-low)
43         .equ   BTN_DEBOUNCE = 50   ; Button debounce time in milliseconds
44
45 ;-----
46 ; SRAM ALLOCATION - System variables
47 ;-----
48 .dseg
49 flags:    .byte 1           ; System flags (bit 0 = temperature ready)
```

```

50 temp_lsb:    .byte 1          ; DS18B20 temperature LSB
51 temp_msb:    .byte 1          ; DS18B20 temperature MSB
52 phase:       .byte 1          ; Temperature sensor phase (0=convert, 1=read)
53
54 ;-----
55 ;  CODE SEGMENT START
56 ;-----
57 .cseg
58
59 ;=====
60 ;  INTERRUPT VECTORS - Defines system interrupt handlers
61 ;=====
62     .org 0
63     jmp reset      ; Reset vector - system startup
64
65     .org INT0addr   ; Next state button (increment state)
66     jmp int0_isr
67     .org INT1addr   ; Previous state button (decrement state)
68     jmp int1_isr
69     .org INT2addr   ; Home button (return to home state)
70     jmp int2_isr
71     .org INT3addr   ; Doctor mode button (diagnostic mode)
72     jmp int3_isr
73     .org OVF1addr   ; Timer-1 overflow (1 second tick)
74     jmp t1_isr
75
76 ;-----
77 ;  LIBRARY INCLUDES - External code modules
78 ;-----
79     ; User interface and IO modules
80     .include "lcd.asm"        ; LCD display driver
81     .include "printf.asm"     ; Formatted text output
82
83     ; Sensor and hardware interface modules
84     .include "wire1.asm"      ; 1-Wire protocol for DS18B20
85     .include "ws2812_driver.asm" ; WS2812 RGB LED driver
86     .include "encoder.asm"    ; Rotary encoder input handler
87     .include "ws2812_helpers.asm" ; WS2812 utility functions
88
89 ;-----
90 ;  STATE MODULES - Application state implementations
91 ;-----
92     ; Each state module implements a different console mode
93     .include "home_state.asm"  ; Home screen and temperature display
94     .include "snake_state.asm"  ; Snake game implementation
95     .include "game2_state.asm"  ; Second game implementation
96     .include "game3_state.asm"  ; Third game implementation
97     .include "doctor_state.asm" ; Diagnostic/doctor mode
98
99 ;=====
100 ;  SYSTEM INITIALIZATION - Hardware and peripherals setup

```

```

101 ;=====
102 reset:
103         ; Initialize stack and core peripherals
104 LDSP  RAMEND          ; Set stack pointer to top of RAM
105 rcall LCD_init          ; Initialize LCD display
106 rcall wire1_init        ; Initialize 1-Wire interface
107 rcall encoder_init      ; Initialize rotary encoder
108
109         ; Configure buttons (PD0–PD3) as inputs with pull-ups
110 cbi  DDRD,0            ; Set as input (next state)
111 cbi  DDRD,1            ; Set as input (prev state)
112 cbi  DDRD,2            ; Set as input (home)
113 cbi  DDRD,3            ; Set as input (doctor mode)
114 sbi  PORTD,0           ; Enable pull-up
115 sbi  PORTD,1           ; Enable pull-up
116 sbi  PORTD,2           ; Enable pull-up
117 sbi  PORTD,3           ; Enable pull-up
118
119         ; Initialize WS2812 RGB LED matrix
120 rcall ws_init           ; Sets PD7 as output for LED data
121
122         ; Configure heartbeat LED on PF7 (active-low)
123 OUTEI DDRF,(1<<LED_BIT)    ; Set PF7 as output
124 OUTEI PORTF,(1<<LED_BIT)    ; Turn LED off initially
125
126         ; Ensure WS2812 line is idle low until driver activates
127 cbi  PORTD,7
128
129         ; Configure Timer-1 for 1-second periodic interrupt
130 ldi  w,T1_PREH          ; Load high byte of preload value
131 out  TCNT1H,w
132 ldi  w,T1_PREL          ; Load low byte of preload value
133 out  TCNT1L,w
134 ldi  w,(1<<CS12)|(1<<CS10) ; Set prescaler to clk/1024
135 out  TCCR1B,w
136 OUTI TIMSK,(1<<TOIE1)    ; Enable Timer-1 overflow interrupt
137
138         ; Configure external interrupts (INT0–INT3) for falling edge
139 OUTEI EICRA,0b10101010    ; Set falling edge for all interrupts
140 OUTI EIMSK,0b00001111    ; Enable INT0–INT3
141
142 sei                  ; Enable global interrupts
143
144         ; Initialize temperature sensing
145 clr  w                  ; Set phase to 0 (conversion mode)
146 sts  phase,w
147 rcall temp_convert       ; Start first temperature conversion
148
149     clr  sel              ; Start in Home state (ST_HOME)
150
151 ;=====

```

```

152 ; MAIN PROGRAM LOOP – State machine implementation
153 ;=====
154 main_loop:
155 switch:
156         ; Copy current state to stable register for comparison
157         mov s,sel
158
159         ; State dispatch – select appropriate handler based on current state
160         _CPI s,ST_HOME
161         brne swSnake           ; If not HOME state, check next state
162         rcall home_init        ; Initialize HOME state
163         rjmp  switch           ; Return to state check
164
165 swSnake:   ; Snake game state handler
166         _CPI s,ST_GAME1
167         brne swGameTwo        ; If not GAME1 state, check next state
168         rcall snake_game_init ; Initialize Snake game
169         rjmp  switch           ; Return to state check
170
171 swGameTwo: ; Game 2 state handler
172         _CPI s,ST_GAME2
173         brne swGameThree      ; If not GAME2 state, check next state
174         rcall gameTwoInit     ; Initialize Game 2
175         rjmp  switch           ; Return to state check
176
177 swGameThree: ; Game 3 state handler
178         _CPI s,ST_GAME3
179         brne swDoctor          ; If not GAME3 state, must be DOCTOR
state
180         rcall gameThreeInit    ; Initialize Game 3
181         rjmp  switch           ; Return to state check
182
183 swDoctor:  ; Doctor/diagnostic mode handler
184         rcall doctorInit       ; Initialize Doctor mode
185         rjmp  switch           ; Return to state check
186
187 ;=====
188 ; TEMPERATURE SENSING – DS18B20 interface functions
189 ;=====
190 ; These routines handle the temperature sensor communication
191 ; using the 1-Wire protocol with the DS18B20 sensor.
192 ;
193
194 ; Initiates a temperature conversion on the DS18B20 sensor
195 temp_convert:
196         push s                 ; Save scratch register
197         rcall wire1_reset      ; Reset 1-Wire bus
198         ldi a0,skipROM         ; Skip ROM command (address all devices)
199         rcall wire1_write
200         ldi a0,convertT        ; Start temperature conversion
201         rcall wire1_write

```

```

202         pop   s          ; Restore scratch register
203         ret
204
205 ; Reads temperature data from the DS18B20 sensor
206 temp_fetch:
207         push  s          ; Save scratch register
208         rcall wire1_reset    ; Reset 1-Wire bus
209         ldi   a0,skipROM      ; Skip ROM command
210         rcall wire1_write
211         ldi   a0,readScratchpad ; Read scratchpad command
212         rcall wire1_write
213         rcall wire1_read       ; Read LSB of temperature
214         sts   temp_lsb,a0
215         rcall wire1_read       ; Read MSB of temperature
216         sts   temp_msb,a0
217         pop   s          ; Restore scratch register
218         ret
219
220 ;-----
221 ; TEMPERATURE BACKGROUND TASK – Periodic temperature monitoring
222 ;-----
223 ; This routine handles the background temperature monitoring process
224 ; which alternates between conversion and reading phases.
225 ;-----
226
227 temp_task:
228         ; Clear temperature ready flag
229         lds   s,flags
230         _ANDI s,~(1<<FLG_TEMP) ; Clear temperature ready flag
231         sts   flags,s
232
233         ; Check current phase and handle accordingly
234         lds   s,phase
235         tst   s          ; Check if phase = 0 (convert) or 1 (read)
236         breq  temp_do_convert ; If phase=0, start conversion
237
238         ; Phase 1: Read temperature data
239         rcall temp_fetch        ; Read temperature from sensor
240         clr   s          ; Set phase back to 0 (convert)
241         sts   phase,s
242         ret
243
244 temp_do_convert:
245         ; Phase 0: Start temperature conversion
246         rcall temp_convert     ; Start temperature conversion
247         _LDI  s,1           ; Set phase to 1 (read)
248         sts   phase,s
249         ret
250
251 ;=====
252 ; INTERRUPT SERVICE ROUTINES – Button and timer handlers

```

```

253 ;=====
254
255 ;----- INT0 - Next state button -----
256 int0_isr:
257     push w                      ; Save working register
258     inc sel                     ; Increment state
259     ldi w,REG_STATES           ; Load maximum state number
260     cp sel,w                  ; Compare current state with max
261     brlo int0_done             ; If less, we're good
262     clr sel                   ; Otherwise wrap around to 0
263 int0_done:
264     ; WAIT_MS BTN_DEBOUNCE      ; Optional: Add debounce delay
265     pop w                      ; Restore working register
266     reti                       ; Return from interrupt
267
268 ;----- INT1 - Previous state button -----
269 int1_isr:
270     push w                      ; Save working register
271     tst sel                     ; Test if current state is 0
272     brne int1_dec               ; If not 0, simply decrement
273     ldi w,REG_STATES-1         ; Otherwise wrap to highest state
274     mov sel,w                  ; Set state to previous
275     ; WAIT_MS BTN_DEBOUNCE      ; Optional: Add debounce delay
276     pop w                      ; Restore working register
277     reti
278 int1_dec:
279     dec sel                     ; Decrement state
280     pop w                      ; Restore working register
281     ; WAIT_MS BTN_DEBOUNCE      ; Optional: Add debounce delay
282     reti
283
284 ;----- INT2 - Home button -----
285 int2_isr:
286     clr sel                     ; Set state to home (0)
287     reti                         ; Return from interrupt
288
289 ;----- INT3 - Doctor mode button -----
290 int3_isr:
291     push w                      ; Save working register
292     ldi w,ST_DOCTOR              ; Load doctor mode state number
293     mov sel,w                  ; Set current state to doctor mode
294     pop w                      ; Restore working register
295     reti                         ; Return from interrupt
296
297 ;----- Timer-1 overflow (1 second tick) -----
298 t1_isr:
299     push w                      ; Save working register
300     push _w                     ; Save ISR scratch register
301
302     ; Reload timer for next 1-second interval
303     ldi w,T1_PREH                ; Load high byte of preload

```

```
304     out    TCNT1H,w
305     ldi    w,T1_PREL           ; Load low byte of preload
306     out    TCNT1L,w
307
308     ; Toggle heartbeat LED on PF7
309     lds    s,PORTF            ; Get current port state
310     ldi    _w,(1<<LED_BIT)   ; Prepare bit mask
311     eor    s,_w              ; Toggle LED bit
312     sts    PORTF,s          ; Update port
313
314     ; Set temperature task flag for background processing
315     lds    s,flags            ; Get current flags
316     _ORI   s,(1<<FLG_TEMP)   ; Set temperature ready flag
317     sts    flags,s          ; Update flags
318
319     pop    _w                ; Restore ISR scratch register
320     pop    w                 ; Restore working register
321     reti                   ; Return from interrupt
322 =====
```

medical_thermo/medical_thermo/home_state.asm

```
1 ;=====
2 ; HOME STATE - Main Menu Interface
3 ;=====
4 ; Target: ATmega128L @ 4MHz
5 ;
6 ; Description:
7 ; This file implements the HOME state, which serves as the main menu
8 ; for the medical console. It displays "HOME" on the LCD screen and
9 ; creates a green pattern with strategic "holes" (blank pixels) on the
10 ; 8x8 RGB LED matrix to create a recognizable visual pattern.
11 ;
12 ; Functions:
13 ; - home_init: Initialize the HOME state (LCD and LED matrix)
14 ; - home_wait: Main polling loop that monitors the current state
15 ; - matrix_holes: Helper function to create the patterned LED display
16 ;
17 ; Register Usage:
18 ; - s: Used to check current system state
19 ; - a0-a2: RGB color components for LED matrix (GRB order)
20 ; - r22: Counter for filling the buffer
21 ; - r24, r25: X,Y coordinates for pixel addressing
22 ; - Z: Memory pointer for frame buffer access
23 ;
24 ; Dependencies:
25 ; - Requires LCD driver for text display
26 ; - Uses WS2812 driver for LED matrix control
27 ; - Relies on ST_HOME constant from main.asm
28 ;
29 ; Last Modified: May 25, 2025
30 ;=====

31 ;
32 ;-----
33 ; HOME state - draw green background with "holes" off(smiley face)
34 ;-----

35
36 home_init:
37     rcall    lcd_clear
38     PRINTF   LCD
39     .db      "HOME", 0, 0
40
41     ; prepare green for fill (GRB = 0x0F,0x00,0x00)
42     ldi      a0, 0x0F      ; G
43     ldi      a1, 0x00      ; R
44     ldi      a2, 0x00      ; B
45     rcall    matrix_holes
46
47 ;-----
48 ; MAIN LOOP - Wait until user changes state
49 ;-----
```

```

50  home_wait:
51      mov      s, sel
52      _CPI    s, ST_HOME
53      brne   home_done
54      WAIT_MS 50
55      rjmp   home_wait
56
57 ;-----
58 ;  CLEANUP – Return to main state handler
59 ;-----
60 home_done:
61     ret
62
63
64 =====
65 ;  matrix_holes – fill 8x8 buffer green, then blank specified pixels
66 ;  clobbers: a0–a2, r0, r22, r24–r25, Z, w=r16 (all scratch)
67 =====
68 matrix_holes:
69     push    r22
70     push    ZL
71     push    ZH
72
73 ;-----
74 ;  BUFFER PREPARATION – Fill entire matrix with green color
75 ;-----
76     ldi    ZL, low(WS_BUF_BASE)
77     ldi    ZH, high(WS_BUF_BASE)
78     ldi    r22, 64
79 mh_fill:
80     st     Z+, a0
81     st     Z+, a1
82     st     Z+, a2
83     dec   r22
84     brne  mh_fill
85
86 ;-----
87 ;  PATTERN CREATION – Turn off specific pixels to create pattern
88 ;-----
89 ; (1,1)
90     ldi    r24,1
91     ldi    r25,1
92     rcall ws_idx_xy
93     rcall ws_offset_idx
94     st    Z+, r1
95     st    Z+, r1
96     st    Z,  r1
97
98 ; (2,1)
99     ldi    r24,2
100    ldi    r25,1

```

```

101      rcall  ws_idx_xy
102      rcall  ws_offset_idx
103      st     Z+, r1
104      st     Z+, r1
105      st     Z,   r1
106
107      ; (5,1)
108      ldi   r24,5
109      ldi   r25,1
110      rcall ws_idx_xy
111      rcall ws_offset_idx
112      st    Z+, r1
113      st    Z+, r1
114      st    Z,   r1
115
116      ; (6,1)
117      ldi   r24,6
118      ldi   r25,1
119      rcall ws_idx_xy
120      rcall ws_offset_idx
121      st    Z+, r1
122      st    Z+, r1
123      st    Z,   r1
124
125      ; (1,2)
126      ldi   r24,1
127      ldi   r25,2
128      rcall ws_idx_xy
129      rcall ws_offset_idx
130      st    Z+, r1
131      st    Z+, r1
132      st    Z,   r1
133
134      ; (2,2)
135      ldi   r24,2
136      ldi   r25,2
137      rcall ws_idx_xy
138      rcall ws_offset_idx
139      st    Z+, r1
140      st    Z+, r1
141      st    Z,   r1
142
143      ; (5,2)
144      ldi   r24,5
145      ldi   r25,2
146      rcall ws_idx_xy
147      rcall ws_offset_idx
148      st    Z+, r1
149      st    Z+, r1
150      st    Z,   r1
151

```

```

152      ; (6,2)
153      ldi    r24,6
154      ldi    r25,2
155      rcall ws_idx_xy
156      rcall ws_offset_idx
157      st     Z+, r1
158      st     Z+, r1
159      st     Z,   r1
160
161      ; (1,4)
162      ldi    r24,1
163      ldi    r25,4
164      rcall ws_idx_xy
165      rcall ws_offset_idx
166      st     Z+, r1
167      st     Z+, r1
168      st     Z,   r1
169
170      ; (6,4)
171      ldi    r24,6
172      ldi    r25,4
173      rcall ws_idx_xy
174      rcall ws_offset_idx
175      st     Z+, r1
176      st     Z+, r1
177      st     Z,   r1
178
179      ; (2,5)
180      ldi    r24,2
181      ldi    r25,5
182      rcall ws_idx_xy
183      rcall ws_offset_idx
184      st     Z+, r1
185      st     Z+, r1
186      st     Z,   r1
187
188      ; (3,5)
189      ldi    r24,3
190      ldi    r25,5
191      rcall ws_idx_xy
192      rcall ws_offset_idx
193      st     Z+, r1
194      st     Z+, r1
195      st     Z,   r1
196
197      ; (4,5)
198      ldi    r24,4
199      ldi    r25,5
200      rcall ws_idx_xy
201      rcall ws_offset_idx
202      st     Z+, r1

```

```

203      st      Z+, r1
204      st      Z,  r1
205
206      ; (5,5)
207      ldi     r24,5
208      ldi     r25,5
209      rcall   ws_idx_xy
210      rcall   ws_offset_idx
211      st      Z+, r1
212      st      Z+, r1
213      st      Z,  r1
214
215      ;-----
216      ; DATA TRANSMISSION – Send pattern to LED matrix
217      ;-----
218      ldi     ZL, low(WS_BUF_BASE)
219      ldi     ZH, high(WS_BUF_BASE)
220      _LDI    r0, 64
221 mh_send:
222      ld     a0, Z+
223      ld     a1, Z+
224      ld     a2, Z+
225      cli
226      rcall ws_byte3wr
227      sei
228      dec    r0
229      brne  mh_send
230      rcall ws_reset
231
232      ;-----
233      ; CLEANUP – Restore saved registers and return
234      ;-----
235      pop    ZH
236      pop    ZL
237      pop    r22
238      ret

```

medical_thermo/medical_thermo/snake_state.asm

```
1 ;=====-----  
2 ; SNAKE GAME IMPLEMENTATION – Classic Snake Game for LED Matrix  
3 ;=====-----  
4 ; Target: ATmega128L @ 4MHz  
5 ;  
6 ; Game Description:  
7 ; This file implements a classic Snake game that runs on an 8x8 RGB LED  
8 ; matrix. The player controls a snake that grows in length each time it  
9 ; eats an apple. The game ends if the snake collides with the walls.  
10;  
11; Features:  
12; – Blue snake head with green body segments  
13; – Red apple that randomly respawns when eaten  
14; – Rotary encoder control with direction queue  
15; – Growing snake length when eating apples  
16; – Wall collision detection (game over condition)  
17;  
18; Technical Implementation:  
19; – Uses WS2812B RGB LED matrix for display  
20; – Snake stored as positions in SRAM buffer  
21; – Circular queue for processing direction changes  
22; – Pseudo-random number generation for apple placement  
23; – Fixed frame rate gameplay (configurable delay)  
24;  
25; Register Usage:  
26; – r18–r26: Used for game state calculations  
27; – ZL,ZH: Memory pointers for accessing snake data  
28; – a0–a2: RGB color components for LED matrix  
29;  
30; Last Modified: May 25, 2025  
31;=====-----  
32;  
33;-----  
34; SYMBOLIC CONSTANTS – Game parameters and configuration  
35;-----  
36 .equ DIR_UPP          = 0  
37 .equ DIR_RIGHTT       = 1  
38 .equ DIR_DOWNN        = 2  
39 .equ DIR_LEFTT         = 3  
40 .equ DIR_INITT        = DIR_RIGHTT  
41  
42 .equ Apple_INIT_POS   = 45 ; (5,5) in 8x8 matrix  
43  
44 .equ SNAKE_INIT_POS1  = 26  
45 .equ SNAKE_INIT_POS2  = 27  
46 .equ SNAKE_INIT_POS3  = 28  
47 .equ SNAKE_INIT_HEAD_IDX = 2  
48 .equ SNAKE_INIT_LEN    = 3  
49
```

```

50 .equ FRAME_DELAY_MS      = 500
51
52 .equ MATRIX_SIZE         = 8
53 .equ GRID_CELLS          = MATRIX_SIZE * MATRIX_SIZE
54 .equ COORD_MASK          = 0x07
55
56 .equ QUEUE_SIZE          = 8
57 .equ QUEUE_MASK           = QUEUE_SIZE - 1
58
59 .equ APPLE_PLACEMENT_TRIES = 8
60
61 .equ EMPTY_CELL           = 0xFF
62 .equ BODY_GREEN            = 0x0F
63 .equ HEAD_BLUE             = 0x0F
64 .equ APPLE_RED              = 0x0F
65
66 ;-----
67 ; SRAM LAYOUT – Game state variables
68 ;-----
69 .dseg
70 snake_body:    .byte GRID_CELLS      ; packed x + MATRIX_SIZE*y
71 head_idx:      .byte 1
72 tail_idx:      .byte 1
73 snake_len:     .byte 1
74
75 direction:     .byte 1      ; DIR_UPP..DIR_LEFTT
76 apple_pos:     .byte 1      ; EMPTY_CELL = no apple
77
78 turn_queue:    .byte QUEUE_SIZE
79 tq_head:       .byte 1
80 tq_tail:       .byte 1
81 .cseg
82
83 ;=====
84 ; INITIALIZATION – Game setup and data initialization
85 ;=====
86 ; This section initializes the game by:
87 ; – Clearing the LCD and displaying "SNAKE"
88 ; – Initializing the rotary encoder
89 ; – Setting up the initial snake data
90 ; – Drawing the initial game state
91 ;
92 snake_game_init:
93     rcall lcd_clear
94     PRINTF LCD
95     .db "SNAKE", 0
96
97     rcall encoder_init
98     rcall snake_init_data
99     rcall snake_draw
100    rjmp snake_wait

```

```

101
102 ;-----
103 ; DATA INITIALIZATION – Setup initial game state
104 ;-----
105 ; Sets up the initial snake position, direction, apple placement,
106 ; and other game parameters. Clears the entire play field and places
107 ; the snake in its starting position.
108 ;-----
109 snake_init_data:
110     ; clear body buffer → EMPTY_CELL
111     ldi ZL, low(snake_body)
112     ldi ZH, high(snake_body)
113     ldi r22, GRID_CELLS
114 clear_body:
115     ldi w, EMPTY_CELL
116     st Z+, w
117     dec r22
118     brne clear_body
119
120     ; seed snake at three positions
121     ldi ZL, low(snake_body)
122     ldi ZH, high(snake_body)
123     ldi w, SNAKE_INIT_POS1
124     st Z+, w
125     ldi w, SNAKE_INIT_POS2
126     st Z+, w
127     ldi w, SNAKE_INIT_POS3
128     st Z , w
129
130     ; indices & length
131     ldi w, SNAKE_INIT_HEAD_IDX
132     sts head_idx, w
133     clr w
134     sts tail_idx, w
135     ldi w, SNAKE_INIT_LEN
136     sts snake_len, w
137
138     ; initial direction & apple
139     ldi w, DIR_INITT
140     sts direction, w
141     ldi w, Apple_INIT_POS
142     sts apple_pos, w
143
144     ; queue pointers
145     clr w
146     sts tq_head, w
147     sts tq_tail, w
148
149     ; encoder state
150     clr a0
151     clr b0

```

```

152     in w, ENCOD
153     sts enc_old, w
154     ret ; Timer-0 prescaler untouched – PRNG read only
155
156 =====
157 ; MAIN LOOP – Game cycle with fixed frame rate
158 =====
159 ; Controls the main game timing loop. Each frame consists of:
160 ; – A fixed delay period (FRAME_DELAY_MS)
161 ; – Processing user input during the delay
162 ; – Moving the snake
163 ; – Drawing the updated game state
164 ; – Checking if player has exited the game
165 ;
166 snake_wait:
167     ; start-of-frame delay
168     ldi r24, low(FRAME_DELAY_MS)
169     ldi r25, high(FRAME_DELAY_MS)
170 frame_delay:
171     rcall update_game
172     WAIT_MS 1
173     sbiw r24, 1
174     brne frame_delay
175
176     rcall move_snake
177     rcall snake_draw
178
179     ; prepare next frame
180     ldi r24, low(FRAME_DELAY_MS)
181     ldi r25, high(FRAME_DELAY_MS)
182     mov s, sel
183     _CPI s, ST_GAME1
184     breq frame_delay
185     ret
186
187 =====
188 ; USER INPUT HANDLING – Process rotary encoder movements
189 =====
190 ; Reads the rotary encoder and updates the snake's direction based
191 ; on encoder rotation. Implements a queue system to store direction
192 ; changes that haven't been processed yet. Prevents 180° turns.
193 ;
194 update_game:
195     push r25
196     push r24
197     push r18
198     push r19
199     push r20
200     push r21
201     push r22
202     push r23

```

```

203
204     rcall encoder_update          ; r15 = ±1 or 0
205     mov  r19, r15
206     tst  r19
207     brne enc_move
208     rjmp enc_exit
209
210 enc_move:
211     lds  r21, direction
212     tst  r19
213     brmi enc_left
214
215 enc_right:
216     mov  r20, r21
217     inc  r20
218     cpi  r20, DIR_LEFTT+1
219     brlo enc_chk
220     clr  r20
221     rjmp enc_chk
222
223 enc_left:
224     mov  r20, r21
225     tst  r20
226     brne enc_left_ok
227     ldi  r20, DIR_LEFTT
228     rjmp enc_chk
229 enc_left_ok:
230     dec  r20
231
232 enc_chk:                      ; reject 180°
233     mov  r22, r21 ; r22 = old_direction
234     subi r22, -2   ; r22 = old_direction + 2
235     andi r22, 0x03 ; r22 = (old_direction + 2) & 0b11 = (old_direction + 2) mod
4
236     cp   r20, r22   ; compare new_direction to the 180°-opposite
237     breq enc_exit   ; if equal, reject the 180° turn
238
239     ; enqueue
240     lds  r23, tq_head
241     mov  r24, r23
242     inc  r24
243     andi r24, QUEUE_MASK
244     lds  r22, tq_tail
245     cp   r24, r22
246     breq enc_exit
247
248     ldi  ZL, low(turn_queue)
249     ldi  ZH, high(turn_queue)
250     add  ZL, r23
251     brcc enc_store
252     inc  ZH

```

```

253 enc_store:
254     st Z, r20
255     sts tq_head, r24
256
257     ; dequeue immediately
258     lds r22, tq_tail
259     lds r23, tq_head
260     cp r22, r23
261     breq enc_exit
262
263     ldi ZL, low(turn_queue)
264     ldi ZH, high(turn_queue)
265     add ZL, r22
266     brcc enc_read
267     inc ZH
268 enc_read:
269     ld r20, Z
270     inc r22
271     andi r22, QUEUE_MASK
272     sts tq_tail, r22
273     sts direction, r20
274
275 enc_exit:
276     pop r23
277     pop r22
278     pop r21
279     pop r20
280     pop r19
281     pop r18
282     pop r24
283     pop r25
284     ret
285
286 =====
287 ; SNAKE MOVEMENT - Update snake position and handle collisions
288 =====
289 ; Calculates the snake's new head position based on current direction
290 ; Checks for collisions with walls and handles apple eating
291 ; Updates the snake's length and position data
292 -----
293 move_snake:
294     lds r18, direction
295
296     ; fetch current head
297     lds r19, head_idx
298     ldi ZL, low(snake_body)
299     ldi ZH, high(snake_body)
300     add ZL, r19
301     brcc head_ptr
302     inc ZH
303 head_ptr:

```

```

304     ld    r20, Z
305
306     ; unpack x,y
307     mov   r21, r20
308     andi r21, COORD_MASK
309     mov   r22, r20
310     lsr   r22
311     lsr   r22
312     lsr   r22
313     andi r22, COORD_MASK
314
315     ; border check & compute next cell
316     cpi  r18, DIR_RIGHTT
317     breq dir_right
318     cpi  r18, DIR_LEFTT
319     breq dir_left
320     cpi  r18, DIR_UPP
321     breq dir_up
322     inc   r22
323     cpi  r22, MATRIX_SIZE
324     brne pack_cell
325     rjmp hit_wall
326
327 dir_right:
328     inc   r21
329     cpi  r21, MATRIX_SIZE
330     brne pack_cell
331     rjmp hit_wall
332
333 dir_left:
334     tst   r21
335     breq hit_wall
336     dec   r21
337     rjmp pack_cell
338
339 dir_up:
340     tst   r22
341     breq hit_wall
342     dec   r22
343
344 pack_cell:
345     ; pack new head
346     mov   r20, r22
347     lsl   r20
348     lsl   r20
349     lsl   r20
350     add   r20, r21
351
352     ; apple collision?
353     lds   r23, apple_pos
354     cpi  r23, EMPTY_CELL

```

```

355     breq write_head
356     cp    r20, r23
357     brne write_head
358
359     ; eat apple → reposition & grow
360     ldi   r23, EMPTY_CELL
361     sts   apple_pos, r23
362     rcall place_new_apple
363     lds   r24, snake_len
364     cpi   r24, GRID_CELLS
365     breq write_head
366     inc   r24
367     sts   snake_len, r24
368     rjmp  write_head_no_tail
369
370 write_head:
371     ; advance tail normally
372     lds   r21, tail_idx
373     inc   r21
374     cpi   r21, GRID_CELLS
375     brlo  tail_ok
376     clr   r21
377 tail_ok:
378     sts   tail_idx, r21
379
380 write_head_no_tail:
381     ; advance head index & write new head
382     lds   r19, head_idx
383     inc   r19
384     cpi   r19, GRID_CELLS
385     brlo  idx_ok_write
386     clr   r19
387 idx_ok_write:
388     sts   head_idx, r19
389     ldi   ZL, low(snake_body)
390     ldi   ZH, high(snake_body)
391     add   ZL, r19
392     brcc  write_ptr
393     inc   ZH
394 write_ptr:
395     st    Z, r20
396     ret
397
398 hit_wall:
399     rcall lcd_clear
400     PRINTF LCD
401     .db "GAME OVER", 0
402 freeze_game:
403     mov   r18, sel
404     _CPI r18, ST_GAME1
405     breq freeze_game

```

```

406     ret
407
408 ;=====
409 ; APPLE PLACEMENT – Generate random position for new apple
410 ;=====
411 ; Places a new apple at a random position that doesn't overlap with
412 ; the snake. Uses a pseudo-random number generator with a fixed
413 ; number of placement attempts for consistent timing.
414 ;-----
415 place_new_apple:
416     push r26
417     push r25
418     push r24
419     push r23
420     push r22
421     push r21
422     push r20
423     push r19
424     push r18
425
426     clr  r18          ; first free candidate marker
427     ldi  r23, APPLE_PLACEMENT_TRIES
428
429 loop_iter:
430     ; PRNG candidate
431     in   r24, TCNT0
432     in   r26, ADCL
433     eor  r24, r26
434     lds  r19, head_idx
435     add  r24, r19
436
437     mov  r25, r24
438     andi r24, COORD_MASK
439     lsr  r25
440     lsr  r25
441     lsr  r25
442     andi r25, COORD_MASK
443
444     mov  r20, r25
445     lsl  r20
446     lsl  r20
447     lsl  r20
448     add  r20, r24
449
450     ; compare against all snake segments
451     lds  r22, tail_idx
452     lds  r24, snake_len
453     clr  r21          ; i = 0
454 scan_loop:
455     cp   r21, r24
456     breq free_found

```

```

457     mov    _w, r22
458     add    _w, r21
459     cpi    _w, GRID_CELLS
460     brlo  idx_ok
461     subi   _w, GRID_CELLS
462 idx_ok:
463     ldi    ZL, low(snake_body)
464     ldi    ZH, high(snake_body)
465     add    ZL, _w
466     brcc  buf_ptr
467     inc    ZH
468 buf_ptr:
469     ld     _w, Z
470     cp     _w, r20
471     breq  clash_found
472     inc    r21
473     rjmp  scan_loop
474
475 clash_found:
476     ; segment clash → skip storing
477     rjmp  loop_continue
478
479 free_found:
480     tst   r18
481     brne loop_continue
482     mov   r18, r20
483
484 loop_continue:
485     dec   r23
486     brne loop_iter
487
488     ; commit result
489     tst   r18
490     brne store_ok
491     mov   r18, r20
492 store_ok:
493     sts   apple_pos, r18
494
495     pop   r18
496     pop   r19
497     pop   r20
498     pop   r21
499     pop   r22
500     pop   r23
501     pop   r24
502     pop   r25
503     pop   r26
504     ret
505
506 ;=====
507 ;  RENDERING – Draw the snake and apple on the LED matrix

```

```

508 ;=====
509 ; Renders the current game state to the LED matrix:
510 ; - Draws the snake body in green
511 ; - Draws the snake head in blue
512 ; - Draws the apple in red
513 ; - Transmits the frame buffer to the physical LED matrix
514 ;-----
515 snake_draw:
516     clr a0
517     clr a1
518     clr a2
519     rcall matrix_solid
520
521     lds r23, tail_idx
522     lds s, snake_len
523     ldi r22, 0
524 draw_loop:
525     cp r22, s
526     breq draw_done
527     mov r24, r23
528     add r24, r22
529     cpi r24, GRID_CELLS
530     brlo idx_ok3
531     subi r24, GRID_CELLS
532 idx_ok3:
533     ldi ZL, low(snake_body)
534     ldi ZH, high(snake_body)
535     add ZL, r24
536     brcc buf_ok3
537     inc ZH
538 buf_ok3:
539     ld w, Z
540     mov r24, w
541     andi r24, COORD_MASK
542     mov r25, w
543     lsr r25
544     lsr r25
545     lsr r25
546     andi r25, COORD_MASK
547     rcall ws_idx_xy
548     rcall ws_offset_idx
549     mov _w, s
550     dec _w
551     cp r22, _w
552     breq head_pix
553 body_pix:
554     ldi a0, BODY_GREEN
555     clr a1
556     clr a2
557     rjmp store_px
558 head_pix:

```

```

559    clr  a0
560    clr  a1
561    ldi  a2, HEAD_BLUE
562    store_px:
563        st   Z+, a0
564        st   Z+, a1
565        st   Z , a2
566        inc  r22
567        rjmp draw_loop
568
569    draw_done:
570        ; apple
571        lds  w, apple_pos
572        cpi  w, EMPTY_CELL
573        breq flush_frame
574        mov  r24, w
575        andi r24, COORD_MASK
576        mov  r25, w
577        lsr   r25
578        lsr   r25
579        lsr   r25
580        andi r25, COORD_MASK
581        rcall ws_idx_xy
582        rcall ws_offset_idx
583        clr  a0
584        ldi  a1, APPLE_RED
585        clr  a2
586        st   Z+, a0
587        st   Z+, a1
588        st   Z , a2
589
590    flush_frame:
591        ldi  ZL, low(WS_BUF_BASE)
592        ldi  ZH, high(WS_BUF_BASE)
593        _LDI r0, GRID_CELLS
594    flush_loop:
595        ld   a0, Z+
596        ld   a1, Z+
597        ld   a2, Z+
598        cli
599        rcall ws_byte3wr
600        sei
601        dec  r0
602        brne flush_loop
603        rcall ws_reset
604        ret

```

medical_thermo/medical_thermo/doctor_state.asm

```
1 ;=====
2 ; DOCTOR MODE – Medical Diagnostic State (ST_DOCTOR)
3 ;=====
4 ; Purpose:
5 ; – Implements a diagnostic mode with temperature display
6 ; – Shows a red Swiss cross on the 8x8 RGB LED matrix
7 ; – Provides continuous temperature monitoring at 4Hz update rate
8 ;
9 ; Functions:
10 ; – doctorInit: Initialize doctor mode (LCD and LED matrix)
11 ; – doctor_loop: Main refresh loop for temperature monitoring
12 ; – matrix_doctor: Draw Swiss cross pattern on LED matrix
13 ;
14 ; Register Usage:
15 ; – r18: Saves caller's a0, temporary for flags and temperature
16 ; – a0, a1: Used for temperature values (LSB/MSB)
17 ; – a0-a2, r0-r1, r22-r25, Z: Used for LED matrix operations
18 ;
19 ; Dependencies:
20 ; – Requires LCD driver for text display
21 ; – Requires WS2812 drivers for LED matrix control
22 ; – Uses temperature sensor via temp_task function
23 ; – Uses PRINTF macro for formatted temperature output
24 ;=====
25
26 doctorInit:
27     push    r18                      ; save caller's a0 slot
28
29     ;-----
30     ; INITIALIZATION – Clear LCD and show doctor mode header
31     ;-
32     rcall   lcd_clear
33     PRINTF  LCD
34     .db      "Doctor",0,0
35
36     ;-
37     ; VISUAL INDICATOR – Display red Swiss cross on LED matrix
38     ;-
39     rcall   matrix_doctor           ; fills buffer & transmits
40
41 ;=====
42 ; MAIN LOOP – Temperature monitoring with 4Hz refresh rate
43 ;=====
44 doctor_loop:
45     ;-
46     ; TEMPERATURE PROCESSING – Handle sensor conversion and reading
47     ;-
48     lds     r18, flags
49     sbrc   r18, FLG_TEMP
```

```

50         rcall    temp_task
51
52         ;-----
53         ; DISPLAY UPDATE – Show current temperature reading on LCD
54         ;-----
55         lds      a0, temp_lsb
56         lds      a1, temp_msb
57         rcall    lcd_clear
58         PRINTF   LCD
59         .db      "Doctor ", FFRAC2+FSIGN, a, 4, $42, "C", 0,0
60
61         ;-----
62         ; TIMING CONTROL – Maintain 4Hz update rate (250ms per cycle)
63         ;-----
64         WAIT_MS 250           ; ≈4 Hz update
65
66         ;-----
67         ; STATE CHECKING – Exit when no longer in Doctor mode
68         ;-----
69         mov      r18, sel
70         cpi      r18, ST_DOCTOR
71         breq    doctor_loop
72
73         ;-----
74         ; CLEANUP – Restore saved register and return to caller
75         ;-----
76         pop      r18
77         ret
78
79
80 =====
81 ; MATRIX_DOCTOR – Create Swiss cross pattern on LED matrix
82 =====
83 ; Description:
84 ; – Fills the entire 8x8 matrix with red color
85 ; – Blanks out specific pixels to form a Swiss cross pattern
86 ; – Transmits the resulting pattern to the LED matrix
87 ;
88 ; Register Usage:
89 ; – a0-a2: RGB color components (G,R,B in WS2812B order)
90 ; – r1: Zero value for blanking pixels
91 ; – r22: Counter for filling the buffer
92 ; – r24, r25: X,Y coordinates for pixel addressing
93 ; – Z: Memory pointer for frame buffer access
94 =====
95 matrix_doctor:
96         push    r22
97         push    ZL
98         push    ZH
99
100        ;-----

```

```

101     ; COLOR SETUP - Set fill color to red (GRB format)
102     ;
103     ldi    a0, 0x00          ; G
104     ldi    a1, 0xF           ; R
105     ldi    a2, 0x00          ; B
106
107     ;
108     ; BUFFER FILLING - Fill entire matrix with red color
109     ;
110     ldi    ZL, low(WS_BUF_BASE)
111     ldi    ZH, high(WS_BUF_BASE)
112     ldi    r22, 64
113     md_fill:
114     st     Z+, a0
115     st     Z+, a1
116     st     Z+, a2
117     dec   r22
118     brne md_fill
119
120     ;
121     ; CROSS PATTERN - Blank out pixels to form Swiss cross shape
122     ;
123     ; list of pixels to turn off:
124     ; (3,1),(4,1),(3,2),(4,2),(3,3),(4,3),(3,4),(4,4),
125     ; (3,5),(4,5),(3,6),(4,6),
126     ; (1,3),(1,4),(2,3),(2,4),(5,3),(5,4),(6,3),(6,4)
127
128     ; helper: a zero color in r1
129     clr   r1
130
131     ; (3,1)
132     ldi    r24,3
133     ldi    r25,1
134     rcall ws_idx_xy
135     rcall ws_offset_idx
136     st     Z+, r1
137     st     Z+, r1
138     st     Z,   r1
139
140     ; (4,1)
141     ldi    r24,4
142     ldi    r25,1
143     rcall ws_idx_xy
144     rcall ws_offset_idx
145     st     Z+, r1
146     st     Z+, r1
147     st     Z,   r1
148
149     ; (3,2)
150     ldi    r24,3
151     ldi    r25,2

```

```
152     rcall  ws_idx_xy
153     rcall  ws_offset_idx
154     st      Z+, r1
155     st      Z+, r1
156     st      Z,   r1
157
158 ; (4,2)
159 ldi    r24,4
160 ldi    r25,2
161 rcall ws_idx_xy
162 rcall ws_offset_idx
163 st    Z+, r1
164 st    Z+, r1
165 st    Z,   r1
166
167 ; (3,3)
168 ldi    r24,3
169 ldi    r25,3
170 rcall ws_idx_xy
171 rcall ws_offset_idx
172 st    Z+, r1
173 st    Z+, r1
174 st    Z,   r1
175
176 ; (4,3)
177 ldi    r24,4
178 ldi    r25,3
179 rcall ws_idx_xy
180 rcall ws_offset_idx
181 st    Z+, r1
182 st    Z+, r1
183 st    Z,   r1
184
185 ; (3,4)
186 ldi    r24,3
187 ldi    r25,4
188 rcall ws_idx_xy
189 rcall ws_offset_idx
190 st    Z+, r1
191 st    Z+, r1
192 st    Z,   r1
193
194 ; (4,4)
195 ldi    r24,4
196 ldi    r25,4
197 rcall ws_idx_xy
198 rcall ws_offset_idx
199 st    Z+, r1
200 st    Z+, r1
201 st    Z,   r1
202
```

```
203      ; (3,5)
204      ldi      r24,3
205      ldi      r25,5
206      rcall   ws_idx_xy
207      rcall   ws_offset_idx
208      st       Z+, r1
209      st       Z+, r1
210      st       Z,   r1
211
212      ; (4,5)
213      ldi      r24,4
214      ldi      r25,5
215      rcall   ws_idx_xy
216      rcall   ws_offset_idx
217      st       Z+, r1
218      st       Z+, r1
219      st       Z,   r1
220
221      ; (3,6)
222      ldi      r24,3
223      ldi      r25,6
224      rcall   ws_idx_xy
225      rcall   ws_offset_idx
226      st       Z+, r1
227      st       Z+, r1
228      st       Z,   r1
229
230      ; (4,6)
231      ldi      r24,4
232      ldi      r25,6
233      rcall   ws_idx_xy
234      rcall   ws_offset_idx
235      st       Z+, r1
236      st       Z+, r1
237      st       Z,   r1
238
239      ; (1,3)
240      ldi      r24,1
241      ldi      r25,3
242      rcall   ws_idx_xy
243      rcall   ws_offset_idx
244      st       Z+, r1
245      st       Z+, r1
246      st       Z,   r1
247
248      ; (1,4)
249      ldi      r24,1
250      ldi      r25,4
251      rcall   ws_idx_xy
252      rcall   ws_offset_idx
253      st       Z+, r1
```

```
254     st      Z+, r1
255     st      Z,   r1
256
257 ; (2,3)
258 ldi    r24,2
259 ldi    r25,3
260 rcall ws_idx_xy
261 rcall ws_offset_idx
262 st    Z+, r1
263 st    Z+, r1
264 st    Z,   r1
265
266 ; (2,4)
267 ldi    r24,2
268 ldi    r25,4
269 rcall ws_idx_xy
270 rcall ws_offset_idx
271 st    Z+, r1
272 st    Z+, r1
273 st    Z,   r1
274
275 ; (5,3)
276 ldi    r24,5
277 ldi    r25,3
278 rcall ws_idx_xy
279 rcall ws_offset_idx
280 st    Z+, r1
281 st    Z+, r1
282 st    Z,   r1
283
284 ; (5,4)
285 ldi    r24,5
286 ldi    r25,4
287 rcall ws_idx_xy
288 rcall ws_offset_idx
289 st    Z+, r1
290 st    Z+, r1
291 st    Z,   r1
292
293 ; (6,3)
294 ldi    r24,6
295 ldi    r25,3
296 rcall ws_idx_xy
297 rcall ws_offset_idx
298 st    Z+, r1
299 st    Z+, r1
300 st    Z,   r1
301
302 ; (6,4)
303 ldi    r24,6
304 ldi    r25,4
```

```
305      rcall  ws_idx_xy
306      rcall  ws_offset_idx
307      st     Z+, r1
308      st     Z+, r1
309      st     Z,   r1
310
311      ;-----
312      ; DATA TRANSMISSION – Send pattern data to LED matrix
313      ;-----
314      ldi    ZL, low(WS_BUF_BASE)
315      ldi    ZH, high(WS_BUF_BASE)
316      _LDI   r0, 64
317  md_send:
318      ld     a0, Z+
319      ld     a1, Z+
320      ld     a2, Z+
321      cli
322      rcall ws_byte3wr
323      sei
324      dec   r0
325      brne md_send
326      rcall ws_reset
327
328      ;-----
329      ; CLEANUP – Restore saved registers and return
330      ;-----
331      pop   ZH
332      pop   ZL
333      pop   r22
334      ret
```

medical_thermo/medical_thermo/game2_state.asm

```
1 ;=====
2 ; GAME 2 STATE - Placeholder for Future Game Implementation
3 ;=====
4 ; Target: ATmega128L @ 4MHz
5 ;
6 ; Description:
7 ; This file implements a placeholder state for a future game (Game 2).
8 ; It currently provides a basic template with a blue LED matrix display
9 ; and "GAME 2" text on the LCD. The implementation loops until the user
10 ; changes to a different state.
11 ;
12 ; Functions:
13 ; - gameTwoInit: Initialize Game 2 state (LCD and LED matrix)
14 ; - game2_wait: Main loop that monitors the current state
15 ;
16 ; Register Usage:
17 ; - s: Used to check current system state
18 ; - a0-a2: RGB color components for LED matrix (GRB order)
19 ; - sel: System state register (defined in main.asm)
20 ;
21 ; Dependencies:
22 ; - Requires LCD driver for text display
23 ; - Uses WS2812 helper functions for LED matrix control
24 ; - Relies on ST_GAME2 constant from main.asm
25 ;
26 ; Last Modified: May 25, 2025
27 ;=====
28
29 ;-----
30 ; GAME 2 state (placeholder for a future game)
31 ;-----
32
33 gameTwoInit:
34 ;-----
35 ; INITIALIZATION - Setup LCD display with game title
36 ;-----
37 rcall lcd_clear
38 PRINTF LCD
39 .db "GAME 2",0,0
40
41 ;-----
42 ; VISUAL INDICATOR - Set LED matrix to solid blue
43 ;-----
44 ; Set color to blue (GRB order: a0=G, a1=R, a2=B)
45 ldi a0, 0x00
46 ldi a1, 0x00
47 ldi a2, 0x0F
48 rcall matrix_solid
49
```

```
50 ;-----  
51 ; MAIN LOOP – Wait until user changes the state  
52 ;-----  
53 game2_wait:  
54 ;-----  
55 ; STATE CHECKING – Monitor if current state is still Game 2  
56 ;-----  
57     mov    s, sel  
58     _CPI    s, ST_GAME2  
59     brne   game2_done          ; Exit if state has changed  
60     WAIT_MS 50                ; Short delay for polling  
61     rjmp   game2_wait         ; Continue waiting  
62  
63 ;-----  
64 ; CLEANUP – Return to main state handler when done  
65 ;-----  
66 game2_done:  
67     ret
```

medical_thermo/medical_thermo/game3_state.asm

```
1 ;=====
2 ; GAME 3 STATE - Placeholder for Future Game Implementation
3 ;=====
4 ; Target: ATmega128L @ 4MHz
5 ;
6 ; Description:
7 ; This file implements a placeholder state for a future game (Game 3).
8 ; It currently provides a basic template with a yellow LED matrix display
9 ; and "GAME 3" text on the LCD. The implementation loops until the user
10 ; changes to a different state.
11 ;
12 ; Functions:
13 ; - gameThreeInit: Initialize Game 3 state (LCD and LED matrix)
14 ; - game3_wait: Main loop that monitors the current state
15 ;
16 ; Register Usage:
17 ; - s: Used to check current system state
18 ; - a0-a2: RGB color components for LED matrix (GRB order)
19 ; - sel: System state register (defined in main.asm)
20 ;
21 ; Dependencies:
22 ; - Requires LCD driver for text display
23 ; - Uses WS2812 helper functions for LED matrix control
24 ; - Relies on ST_GAME3 constant from main.asm
25 ;
26 ; Last Modified: May 25, 2025
27 ;=====

28
29 ;-----
30 ; GAME 3 state
31 ;-----

32
33 gameThreeInit:
34 ;-----
35 ; INITIALIZATION - Setup LCD display with game title
36 ;-----
37 rcall lcd_clear
38 PRINTF LCD
39 .db "GAME 3",0
40
41 ;-----
42 ; VISUAL INDICATOR - Set LED matrix to solid yellow (R+G)
43 ;-----
44 ; Set color to yellow (GRB order: a0=G, a1=R, a2=B)
45 ldi a0, 0x08
46 ldi a1, 0x08
47 ldi a2, 0x00
48 rcall matrix_solid
49
```

```
50 ;-----  
51 ; MAIN LOOP – Wait until user changes the state  
52 ;-----  
53 game3_wait:  
54 ;-----  
55 ; STATE CHECKING – Monitor if current state is still Game 3  
56 ;-----  
57     mov    s, sel  
58     _CPI    s, ST_GAME3  
59     brne   game3_done          ; Exit if state has changed  
60     WAIT_MS 50                ; Short delay for polling  
61     rjmp   game3_wait         ; Continue waiting  
62  
63 ;-----  
64 ; CLEANUP – Return to main state handler when done  
65 ;-----  
66 game3_done:  
67     ret
```