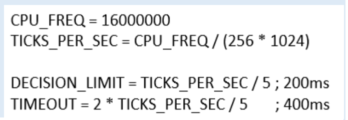
MORSE DECODER

GROUP 18

Jonathan Clarke, Mark Fernandes, Surenthirarajah Sajan, Tabitha Berry

1. Implementation

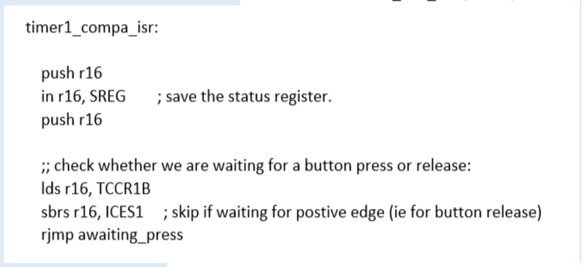
First, we initialized the program and set the timing parameters.



```
CPU_FREQ = 16000000
TICKS_PER_SEC = CPU_FREQ / (256 * 1024)

DECISION_LIMIT = TICKS_PER_SEC / 5 ; 200ms
TIMEOUT = 2 * TICKS_PER_SEC / 5      ; 400ms
```

Then we mapped the alphabet to the 7segment display and set up the output, counter, timer and button registers.

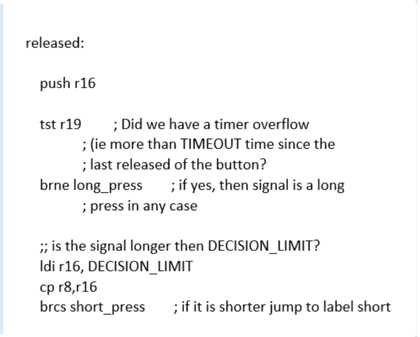
Next, we set up the timer subroutines and a check whether we are waiting for a button press or release.



```
timer1_compa_isr:

  push r16
  in r16, SREG        ; save the status register.
  push r16

  ;; check whether we are waiting for a button press or release:
  lds r16, TCCR1B
  sbrs r16, ICES1     ; skip if waiting for postive edge (ie for button release)
  rjmp awaiting_press
```

The awaiting_release and awaiting_press subroutines then identify what state the button is in and decide whether a letter need be displayed.

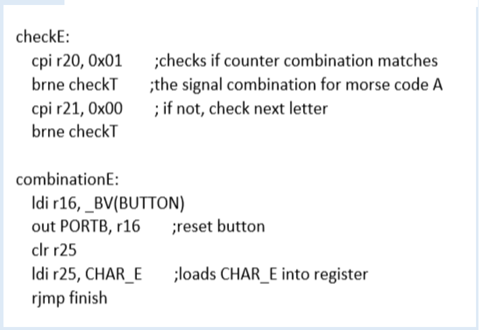
The released subroutine then handles the detection of a Morse signal and identifies whether the signal is a long press (dash) or a short press (dot).



```
released:

  push r16

  tst r19        ; Did we have a timer overflow
                 ; (ie more than TIMEOUT time since the
                 ; last released of the button?
  brne long_press     ; if yes, then signal is a long
                 ; press in any case

  ;; is the signal longer then DECISION_LIMIT?
  ldi r16, DECISION_LIMIT
  cp r8,r16
  brcs short_press      ; if it is shorter jump to label short
```

Whenever a dot or a dash is detected, their corresponding register's bit value is changed to match the signal that is being inputted. The r20 register stores the short press counter and the r21 register stores the long press counter. Eg. For letter "D":dash,dot,dot – r21 = 0001, r20 = 0110

Thereafter the program checks if the counter combination matches the signal combination and thus loads the detected letter or enters error if a letter is not detected.



```
checkE:
  cpi r20, 0x01      ;checks if counter combination matches
  brne checkT       ;the signal combination for morse code A
  cpi r21, 0x00      ; if not, check next letter
  brne checkT

combinationE:
  ldi r16, _BV(BUTTON)
  out PORTB, r16      ;reset button
  clr r25
  ldi r25, CHAR_E      ;loads CHAR_E into register
  rjmp finish
```

Each check subroutine checks how many short presses have been entered and how many long presses have been entered, and in which order each is signaled. The combination subroutine then loads the character into the display and resets the button.

Finally, the finish routine is implemented, we are now waiting for a new signal to be entered, if not, a letter is to be displayed.

Our program implements all characters of the alphabet and includes an error code for incorrect inputs.

2. Functions and Challenges

Our program functions largely as the coursework details outline. The circuitry is exactly as required, with the debounce and timing of button presses and gaps between Morse signals also are as described. An error signal is also outputted by our program when the button is pressed an incorrect amount of times, or when the sequence doesn't correspond to any of our coded letters. Letters are displayed as outlined (we have found a creative work around for certain characters) with a 400ms decision limit between displaying each letter.

We faced some difficulties in the task such as differentiating between characters such as A and N as the Morse sequence of both are mirror images of each other. However, we overcame this by tweaking the code to check the order of inputs and therefore differentiating between the Morse sequence. Additionally, implementing previously taught techniques and adapting them to the current scenario proved challenging but we overcame this through practice and extensive research.

We also faced some circuitry issues due to faulty resistors and found this taxing as it was difficult to identify where the issues in our circuit originated from. Furthermore, inefficient use of wires also caused problems however we solved this by redesigning and using different parts.

Another large difficulty we faced was with the encoder due to it being 2 new programming languages and thus having to essentially translate from one to another caused a lot of confusion. We could've overcome this by more extensive research and learning both languages more thoroughly however due to time constraints this wasn't possible.

3. Contributions

Each member contributed equally throughout the project and played to their strengths.

Jonathan covered much of the coding and trialing different versions of code to produce the final program. Through analysis of previous labs and extensive research, he found the most effective combination of code to efficiently create a Morse decoder. Moreover, he wrote much of the comments for the program and tested the program thoroughly.

Mark did a large portion of research for the decoder and encoder, providing many solutions to problems we faced. His research consisted of reviewing given resources, interpreting labs, through watching YouTube videos and reading articles. Furthermore, he wrote some of the comments for the code to outline its function.

Sajan created the circuitry in the project and wrote the report, providing any solutions to circuit and code problems we faced. He researched AVR assembly through YouTube videos, the working of the Arduino online and analyzing previous labs and given material. Furthermore, he wrote some of the comments for the code to outline its function.