Lab Module

Fundamental of Digital System

Digital Laboratory

Tim Penyusun Modul

Michael Harditya

Muhammad Naufal Faza

Program Studi S1 Teknik Komputer

Fakultas Teknik

Universitas Indonesia

2024

MODUL 1: ASSEMBLY WITH ARDUINO

Pada modul ini, kita akan fokus pada pembelajaran assembly pada Arduino Uno, lebih tepatnya mikrokontroller ATmega328. Assembly adalah bahasa pemrograman tingkat rendah yang memungkinkan kita untuk memanipulasi setiap bit dalam memori dan membuat kode yang sangat efisien dan cepat. Karena assembly memiliki tingkat kontrol yang sangat tinggi, ia memungkinkan kita untuk membuat proyek yang membutuhkan performa tinggi, seperti sistem real-time dan aplikasi yang memerlukan banyak proses matematika.

Dengan mempelajari assembly, kita dapat memahami cara kerja sebuah mikrokontroler pada tingkat dasar dan membuat kode yang sangat efisien dan tepat sasaran. Selain itu, dengan memahami assembly, kita juga dapat memecahkan masalah yang mungkin terjadi dalam pemrograman dengan bahasa pemrograman lainnya.

Namun, perlu dicatat bahwa assembly memiliki learning curve yang cukup tinggi dan membutuhkan waktu dan kesabaran untuk mempelajarinya. Oleh karena itu, modul ini akan memberikan dasar-dasar assembly secara bertahap dan menyediakan contoh-contoh kode untuk membantu mempermudah proses pembelajaran.

Secara keseluruhan, dengan mempelajari assembly pada Arduino Uno, kita dapat memperluas pengetahuan dan keterampilan dalam pemrograman mikrokontroler dan membuat proyek yang lebih kompleks dan menarik.

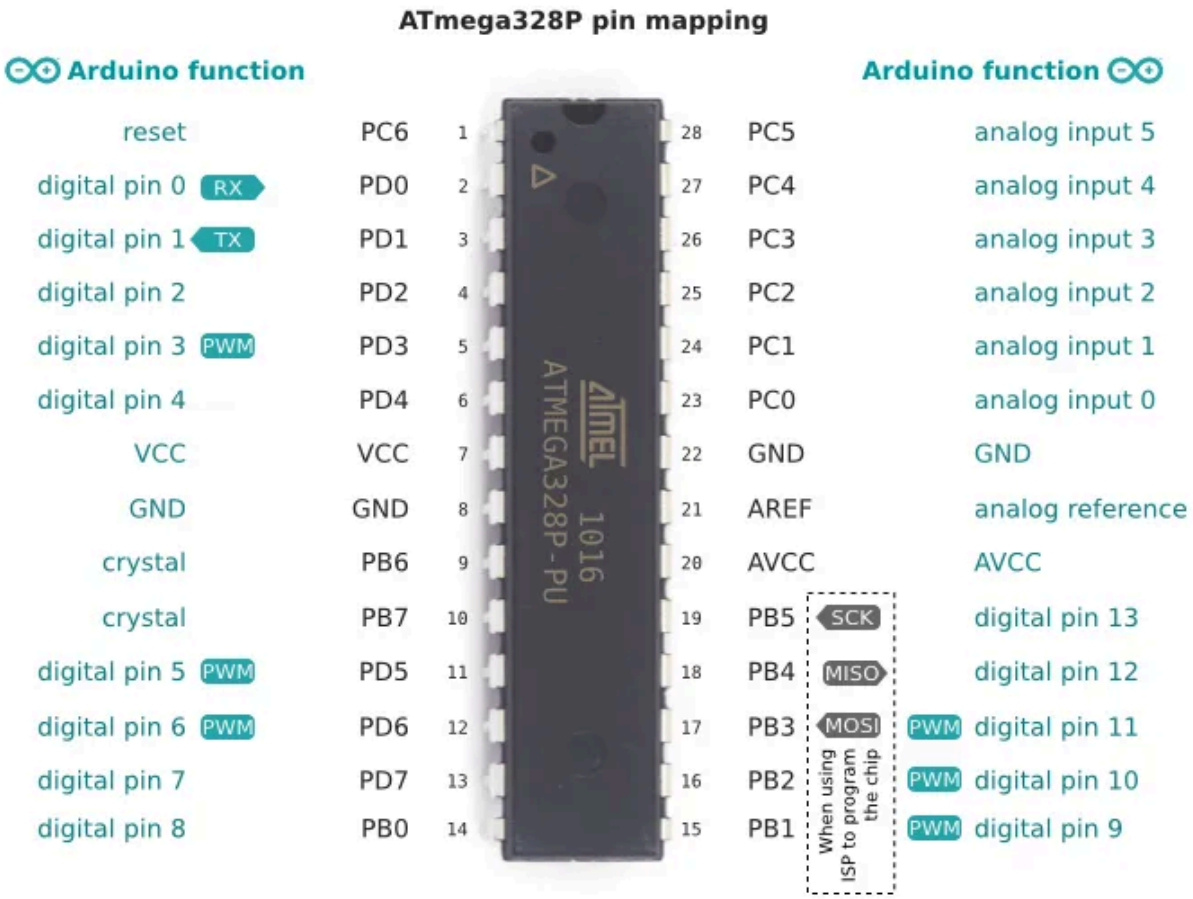
Arduino IDE

Pengetikan dan pengiriman program ke board dapat dilakukan menggunakan tools yaitu **Arduino Integrated Development Environment (IDE)**. Tools ini sudah memuat *library* yang diperlukan oleh Arduino secara tertanam, dan memiliki visualisasi yang mudah dimengerti.

Versi terbaru dari Arduino IDE adalah versi 2.0, dapat diakses melalui Microsoft Store (Windows versi 10 keatas), atau melalui website resmi Arduino (Universal): https://www.arduino.cc/en/software

Dokumentasi dari IDE 2.0 dapat diakses juga pada link berikut: https://docs.arduino.cc/software/ide-v2/tutorials/getting-started-ide-v2

Pengetahuan Hardware Arduino Uno (ATmega328p)



Sumber: http://www.chicoree.fr/w/Arduino_sans_Arduino

Port B, C, dan D adalah port digital input/output pada Arduino Uno. Masing-masing port memiliki 8 bit, sehingga kita dapat mengontrol atau membaca 8 pin digital pada satu port. Port B, C, dan D digunakan untuk mengontrol peralatan elektronik seperti LED, relay, dan lain-lain.

Port B pada Arduino Uno memiliki 8 pin digital, yaitu pin 8 sampai dengan pin 15. Pin 8 sampai dengan pin 13 digunakan untuk mengontrol peralatan elektronik, sedangkan pin 14 sampai dengan pin 15 digunakan untuk komunikasi I2C.

Port D pada Arduino Uno memiliki 8 pin digital, yaitu pin 0 sampai dengan pin 7. Pin 0 sampai dengan pin 7 digunakan untuk mengontrol peralatan elektronik, seperti LED, relay, dan lain-lain. Pin 0 dan 1 digunakan untuk komunikasi serial, sedangkan pin 2 sampai dengan pin 7 digunakan untuk mengontrol peralatan elektronik.

Untuk mengontrol pin digital pada Port B, kita dapat menggunakan instruksi ini:

- DDRB/DDRC/DDRD: Digunakan untuk mengatur pin Port B sebagai input atau output. DDRB/DDRC/DDRD = 0bXXXXXXXX menunjukkan bahwa pin-pin tersebut adalah input, sedangkan DDRB/DDRC/DDRD = 0b11111111 menunjukkan bahwa pin-pin tersebut adalah output.
- PORTB/PORTC/PORTD: Digunakan untuk mengatur nilai logika pada pin Port B. PORTB/PORTC/PORTD = 0bXXXXXXXX menunjukkan bahwa pin-pin tersebut memiliki nilai logika 0, sedangkan PORTB/PORTC/PORTD = 0b11111111 menunjukkan bahwa pin-pin tersebut memiliki nilai logika 1.
- PINB/PINC/PIND: Digunakan untuk membaca nilai logika pada pin Port B. PINB/PINC/PIND = 0bXXXXXXXX menunjukkan bahwa pin-pin tersebut memiliki nilai logika 0, sedangkan PINB/PINC/PIND = 0b11111111 menunjukkan bahwa pin-pin tersebut memiliki nilai logika 1.

Instruction Set

Berikut beberapa instruksi set dari Atmega328p:

1. ADD (Addition)
   Instruksi ADD memungkinkan kita untuk menjumlahkan dua register atau register dan data yang disimpan dalam memori. Contoh sintaks instruksi ADD adalah: ADD R1, R2 atau ADD R1, #data.

2. SUB (Substraction)

Instruksi SUB memungkinkan kita untuk mengurangkan dua register atau register dan data yang disimpan dalam memori. Contoh sintaks instruksi SUB adalah: SUB R1, R2 atau SUB R1, #data.

3. AND (Bitwise AND)
   Instruksi AND memungkinkan kita untuk melakukan operasi logika AND antara dua register atau register dan data yang disimpan dalam memori. Contoh sintaks instruksi AND adalah: AND R1, R2 atau AND R1, #data.

4. OR (Bitwise OR)
   Instruksi OR memungkinkan kita untuk melakukan operasi logika OR antara dua register atau register dan data yang disimpan dalam memori. Contoh sintaks instruksi OR adalah: OR R1, R2 atau OR R1, #data.

5. MUL (Multiplication)
   Instruksi MUL memungkinkan kita untuk melakukan operasi perkalian antara dua register. Contoh sintaks instruksi MUL adalah: MUL R1, R2.

6. JMP (Jump)
   Instruksi JMP memungkinkan kita untuk melakukan jump ke alamat yang ditentukan. Contoh sintaks instruksi JMP adalah: JMP label.

7. RCALL (Relative Call)
   Instruksi RCALL memungkinkan kita untuk memanggil subroutine secara relative, yaitu dengan memasukkan alamat subroutine yang relative terhadap alamat instruksi saat ini. Contoh sintaks instruksi RCALL adalah: RCALL label.

8. RET (Return)
   Instruksi RET memungkinkan kita untuk kembali dari subroutine setelah selesai dieksekusi. Contoh sintaks instruksi RET adalah: RET.

9. CPI (ComPare Immediate)
   Instruksi ini digunakan untuk membandingkan dua bilangan dan memasukkan hasilnya ke dalam register flag. Contoh sintaks instruksi CPI adalah: CPI R0, #0x05.

10. BREQ (Branch if Equal)

Instruksi ini digunakan untuk melakukan branching ke alamat tertentu jika flag equal (Z) bernilai 1. Contoh sintaks instruksi BREQ adalah: BREQ 0x100.

11. MOV (MOVe)
    Instruksi ini digunakan untuk memindahkan data dari satu register ke register lain. Contoh sintaks instruksi MOV adalah: MOV R0, R1.

12. IN (INput)
    Instruksi ini digunakan untuk membaca data dari port I/O tertentu dan memasukkannya ke dalam register. Contoh sintaks instruksi IN adalah: IN R0, 0x10.

13. OUT (OUTput)
    Instruksi ini digunakan untuk menulis data dari register ke port I/O tertentu. Contoh sintaks instruksi OUT adalah: OUT 0x10, R0.

14. LDI (Load Immediate)
    Instruksi ini digunakan untuk memuat data konstan ke dalam register. Contoh sintaks instruksi LDI adalah: LDI R0, #0x10.

15. STS (STore to SRAM)
    Instruksi ini digunakan untuk menyimpan data dari register ke memori SRAM. Contoh sintaks instruksi STS adalah: STS 0x10, R0.

16. ROL (ROtate Left)
    Instruksi ini digunakan untuk melakukan rotasi bit ke kiri. Contoh sintaks instruksi ROL adalah: ROL R0.

17. CBI (Clear Bit in I/O Register)
    Instruksi ini digunakan untuk mengosongkan bit tertentu dalam port I/O. Contoh sintaks instruksi CBI adalah: CBI 0x10, 0x01.

18. CLC (CLear Carry)
    Instruksi ini digunakan untuk mengosongkan bit carry. Contoh sintaks instruksi CLC adalah: CLC.

19. SWAP (SWAP nibbles)
    Instruksi ini digunakan untuk menukar posisi nibble atas dan nibble bawah. Contoh sintaks instruksi SWAP adalah: SWAP R0.

20. ROR (ROtate Right)
    Instruksi ini digunakan untuk memutar bit kekanan pada register atau memori. Contoh sintaks instruksi ROR adalah: ROR R0.

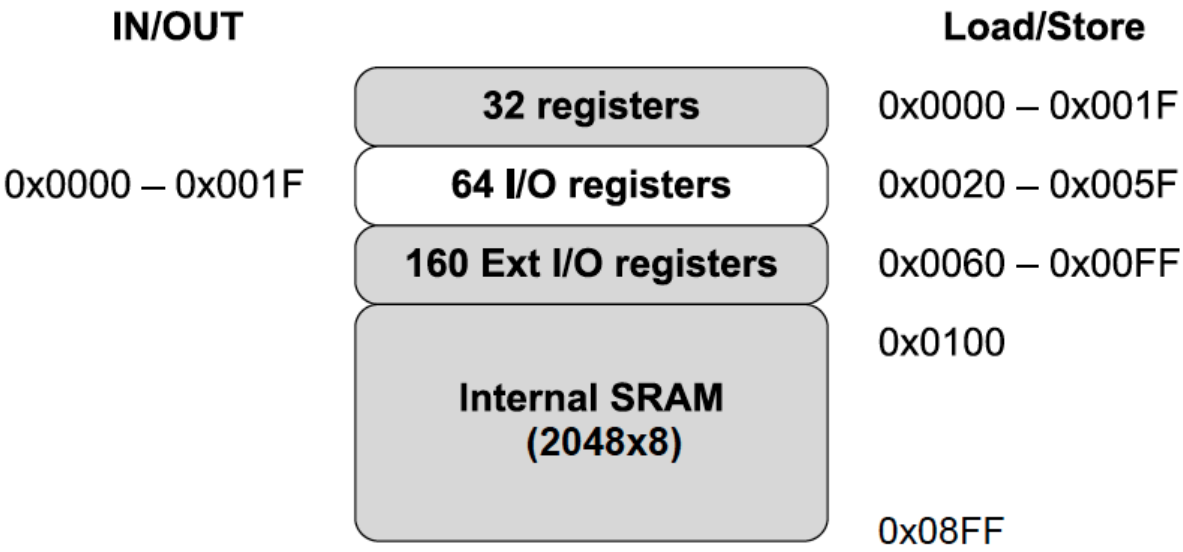
21. BREAK
    Instruksi ini digunakan untuk memaksa program untuk berhenti sementara dan memasuki debugging mode. Contoh sintaks instruksi BREAK adalah: BREAK.

22. NOP (NO Operation)
    Instruksi ini digunakan untuk menunjukkan bahwa tidak ada operasi yang harus dilakukan. Contoh sintaks instruksi NOP adalah: NOP.

23. SLEEP
    Instruksi ini digunakan untuk memasukkan microcontroller ke mode sleep untuk menghemat daya. Contoh sintaks instruksi SLEEP adalah: SLEEP.

24. WDR (WatchDog Timer Reset)
    Instruksi ini digunakan untuk mereset watchdog timer pada microcontroller. Contoh sintaks instruksi WDR adalah: WDR.

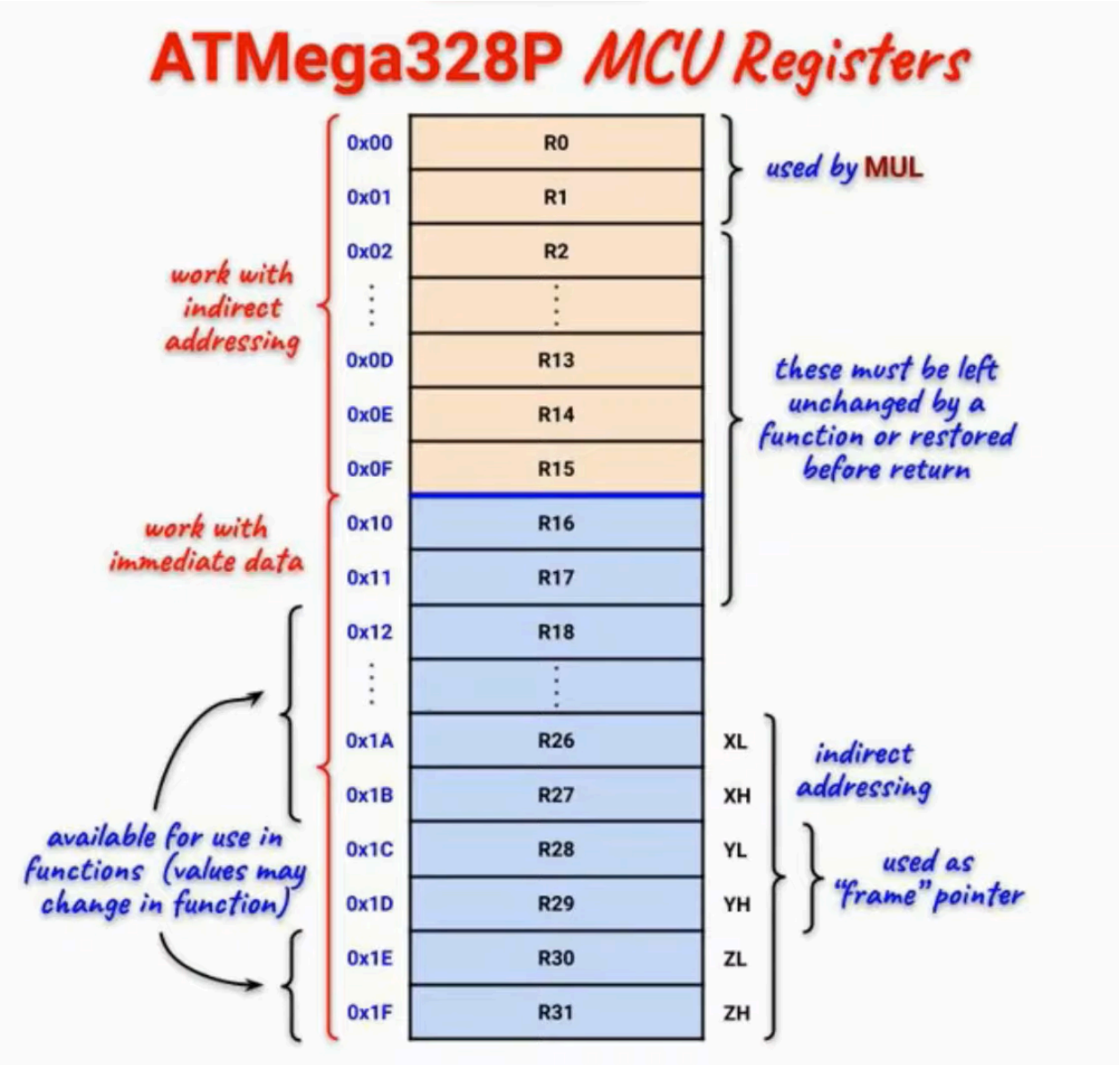
Memory Map



Sumber:
https://www.arnabkumardas.com/blog/wp-content/uploads/2021/04/Atmega328p-ram.png

Memory Map pada ATMega328p adalah bagian yang sangat penting dalam pemrograman Assembly pada Arduino Uno. Memory Map menyediakan informasi tentang bagaimana memori digunakan oleh MCU (Microcontroller Unit) dan bagaimana data dapat ditempatkan dan dibaca dari memori.

32 MCU Registers adalah bagian memori yang digunakan untuk menyimpan data yang digunakan oleh MCU saat melakukan proses. Registers R0 hingga R31 adalah bagian dari memory map yang digunakan sebagai tempat sementara untuk menyimpan data sementara selama proses pemrograman. Register R0 sampai R31 memiliki ukuran 8-bit, sehingga memiliki maksimal nilai 255.



Sumber: Assembly via Arduino (part 2) - Introduction

Untuk menulis kode assembly yang efisien, kita perlu memahami bagaimana cara menggunakan register yang tepat. Register umum dibagi menjadi dua bagian. Bagian pertama adalah register dari R0 hingga R15, sedangkan bagian kedua adalah R16 hingga R31. Register dari R16 hingga R31 lebih serbaguna dan berguna dibandingkan dengan R0 hingga R15, karena register ini bekerja dengan data immediate, yang berarti kita dapat menyimpan byte atau word ke dalam register ini. Untuk menyimpan word, kita harus menggunakan salah satu dari tiga pasangan register X, Y, dan Z. Sedangkan register dari R0 hingga R15 memiliki keterbatasan fungsionalitas, yaitu tidak bisa menyimpan nomor immediate, namun dapat bekerja dengan alamat memori tidak langsung.

SRAM (Static Random Access Memory) adalah memori yang dapat digunakan untuk menyimpan data permanen. SRAM dapat dibaca dan ditulis secara bebas selama MCU sedang beroperasi. SRAM juga memiliki ukuran 8-bit, sehingga memiliki maksimal nilai 255.

160 External I/O Registers adalah bagian memori yang dapat digunakan untuk mengontrol I/O (Input/Output) pada Arduino Uno. Registrasi ini memiliki ukuran 8-bit, sehingga memiliki maksimal nilai 255.

64 I/O Registers adalah bagian memori yang digunakan untuk mengontrol I/O pada Arduino Uno. Register ini memiliki ukuran 8-bit, sehingga memiliki maksimal nilai 255.

Contoh Kode Extern

Berikut adalah contoh kode penggunaan kode assembly untuk mengkustomisasi sebuah fungsi di Arduino.

Kode file.ino:

```
extern "C"{
  void start();
  void led(byte);
}
void setup() {
  start();
}

void loop() {
  led(1);
  led(0);
}
```

Kode file.asm:

```
#define __SFR_OFFSET 0x00
#include "avr/io.h"

.global start
.global led

start:
  SBI DDRB, 4
  RET

led:
  CPI R24, 0x00
  BREQ ledOFF
  SBI PORTB, 4
  RCALL myDelay
  RET

ledOFF:
  CBI PORTB, 4
  RCALL myDelay

.equ delayVal, 10000

myDelay:
  LDI R20, 100
outerLoop:
  LDI R30, lo8(delayVal)
  LDI R31, hi8(delayVal)
  SUBI R20, 1
  BRNE outerLoop
```

Penjelasan:

```
#define __SFR_OFFSET 0x00
#include "avr/io.h"
```

Baris-baris di atas adalah pre-processor directive yang digunakan untuk memasukkan header file "avr/io.h". Header file ini berisi definisi register dalam Atmega328p yang dibutuhkan untuk memprogram pada bahasa assembly.

```
.global start
.global led
```

Baris-baris di atas adalah deklarasi symbol global untuk label "start" dan "led". Ini berarti bahwa label-label tersebut dapat diakses oleh bagian lain dari program.

```
start:
  SBI DDRB, 4
  RET
```

Label "start" adalah bagian dimana program akan memulai eksekusinya. Baris "SBI DDRB, 4" adalah perintah untuk mem-set bit keempat pada register DDRB (Data Direction Register B) sebagai output. Baris "RET" adalah perintah untuk kembali ke lokasi pemanggil.

```
led:
  CPI R24, 0x00
  BREQ ledOFF
  SBI PORTB, 4
  RCALL myDelay
  RET
```

Label "led" adalah bagian dimana program akan membaca status dari LED dan memutuskan apakah LED akan dalam keadaan menyala atau mati. Baris "CPI R24, 0x00" adalah perintah untuk membandingkan nilai dari register R24 dengan nilai 0x00. Jika sama, maka LED akan dalam keadaan mati (jump ke label "ledOFF"). Jika tidak, LED akan dalam keadaan menyala dan baris "SBI PORTB, 4" akan dijalankan untuk mengaktifkan LED. Baris "RCALL myDelay" akan memanggil subroutine "myDelay". Baris "RET" akan mengembalikan program ke lokasi pemanggil.

```
ledOFF:
  CBI PORTB, 4
  RCALL myDelay
```

Label "ledOFF" adalah bagian dimana LED akan dalam keadaan mati. Baris "CBI PORTB, 4" adalah perintah untuk mem-clear bit keempat pada register PORTB sehingga LED akan mati. Baris "RCALL myDelay" akan memanggil subroutine "myDelay".

```
.equ delayVal, 10000
```

```
myDelay:
  LDI R20, 100
outerLoop:
  LDI R30, lo8(delayVal)
  LDI R31, hi8(delayVal)
  SUBI R20, 1
  BRNE outerLoop
```

Baris "RCALL myDelay" adalah pemanggilan subrutin "myDelay", dimana subrutin ini akan melakukan delay dengan waktu yang ditentukan pada konstanta "delayVal". Subrutin "myDelay" akan mengulangi loop sebanyak "R20" kali, setiap loop akan memperlambat waktu dengan nilai "delayVal". Setelah subrutin selesai, program akan kembali ke alamat yang sedang diproses sebelum pemanggilan subrutin.

Baris "LDI R30, lo8(delayVal)" dan "LDI R31, hi8(delayVal)" adalah instruksi untuk memasukkan nilai low byte dan high byte dari konstanta "delayVal" ke dalam register R30 dan R31. Nilai ini digunakan untuk memperlambat waktu dalam subrutin "myDelay".

Baris "SUBI R20, 1" adalah instruksi untuk mengurangi nilai "R20" sebanyak 1 setiap kali loop. Baris "BRNE outerLoop" akan memeriksa apakah "R20" masih bernilai bukan 0. Jika ya, maka program akan kembali ke alamat "outerLoop". Jika tidak, maka subrutin "myDelay" akan selesai dan program akan kembali ke alamat yang sedang diproses sebelum pemanggilan subrutin.

Contoh Kode Blink LED

Langsung .S dan file .ino nya kosong. Dapat disimulasikan di https://wokwi.com/ . Buat pin digital 13 menjadi output LED (atau bisa menggunakan LED Builtin)

```
#define __SFR_OFFSET 0

#include "avr/io.h"

.global main

main:
  sbi   DDRB, 5     ; Set PB5 as output

loop:
  sbi   PORTB, 5     ; Give PB5 high signal
```

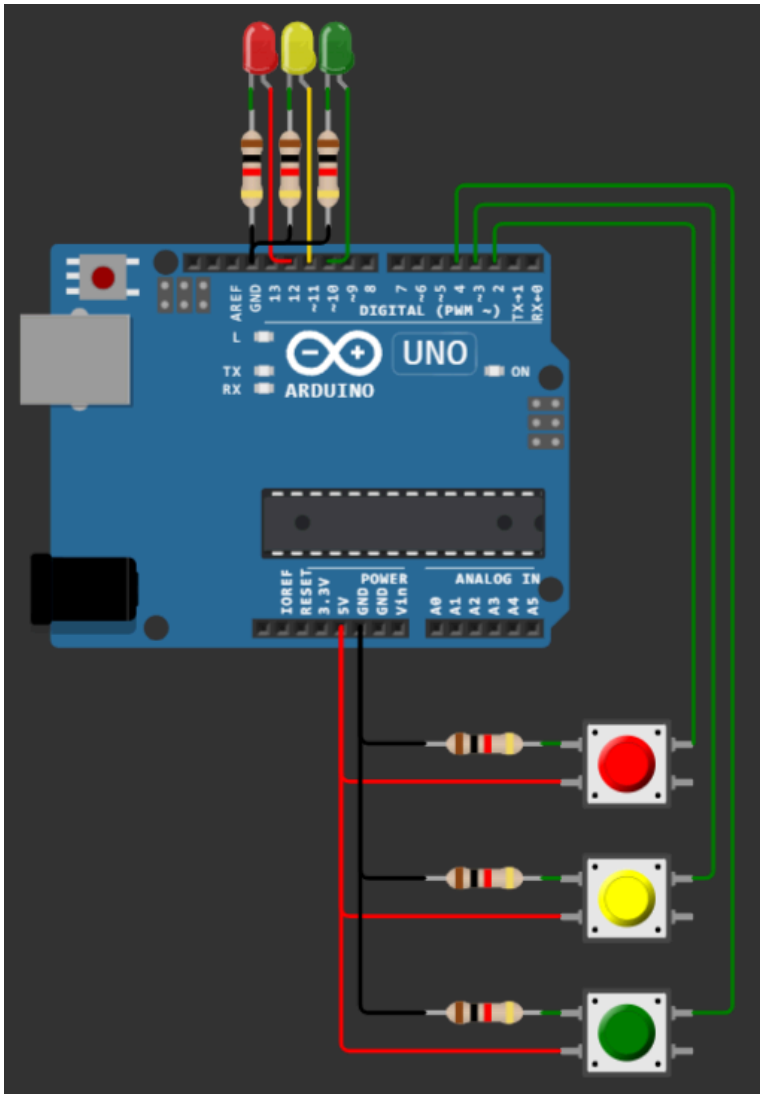
```
  ldi   r25, hi8(1000)  ; Load the upper byte of the 16-bit delay
  ldi   r24, lo8(1000)  ; Load the lower byte of the 16-bit delay
  call  delay_ms      ; Call the delay subroutine
  cbi   PORTB, 5      ; Give PB5 low signals
  ldi   r25, hi8(1000)  ; Load the upper byte of the 16-bit delay
  ldi   r24, lo8(1000)  ; Load the lower byte of the 16-bit delay
  call  delay_ms
  jmp   loop      ; Loop back to the start of the main routine

delay_ms:
    ; Delay about (r25:r24)*ms. Clobbers r30, and r31.
    ; One millisecond is about 16000 cycles at 16MHz.
    ; The inner loop takes 4 cycles, so we repeat it 3000 times
  ldi   r31, hi8(4000)  ; Load the upper byte of the 16-bit delay
  ldi   r30, lo8(4000)  ; Load the lower byte of the 16-bit delay
1:
  sbiw    r30, 1      ; Subtract 1 from the 16-bit delay
  brne    1b      ; If the 16-bit delay is not zero, jump back to the start of
the delay loop
  sbiw    r24, 1      ; Subtract 1 from the 16-bit delay
  brne    delay_ms    ; If the 16-bit delay is not zero, jump back to the start
of the delay subroutine
  ret      ; Return from the subroutine
```

Contoh Kode Input Button

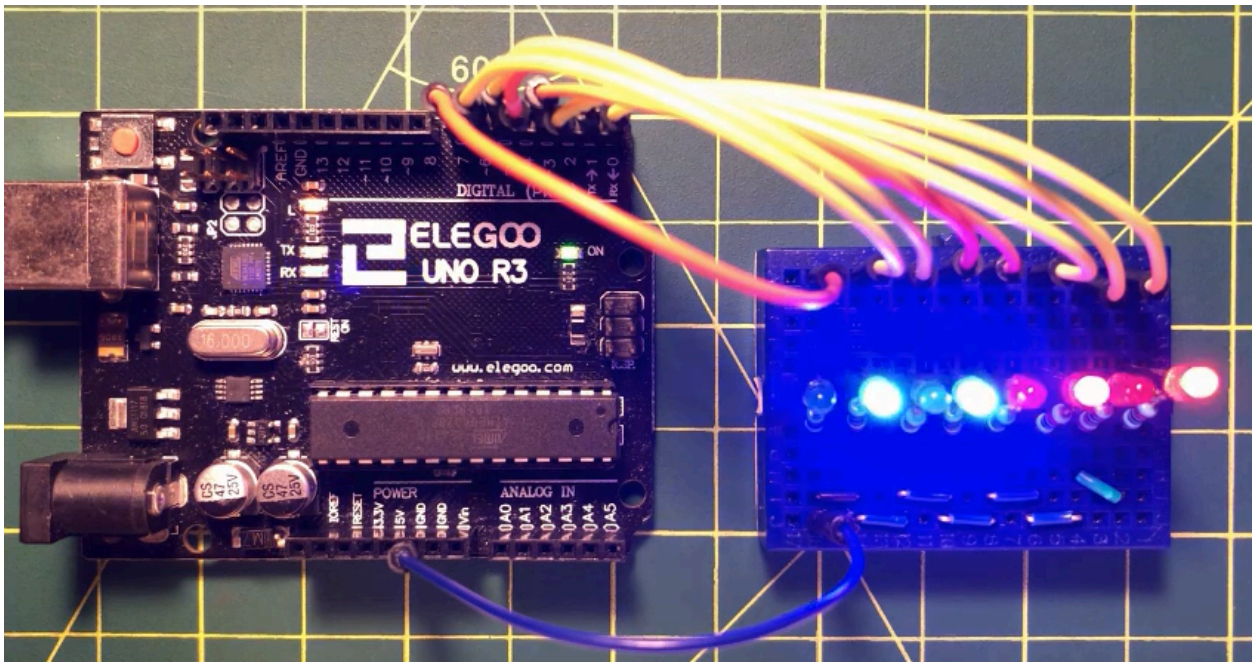
Langsung .S dan file .ino nya kosong. Dapat disimulasikan di https://wokwi.com/ .

```
#define __SFR_OFFSET 0x00
#include "avr/io.h"
;------------------------
.global main

;=============================================================================
main:
    SBI   DDRB, 4             ;set PB4 (pin D12 as o/p - red LED)
    SBI   DDRB, 3             ;set PB3 (pin D11 as o/p - yellow LED)
    SBI   DDRB, 2             ;set PB2 (pin D10 as o/p - green LED)
    CBI   DDRD, 2             ;clear PD2 (pin D02 as i/p - red button)
```

```
    CBI   DDRD, 3             ;clear PD3 (pin D03 as i/p - yellow button)
    CBI   DDRD, 4             ;clear PD4 (pin D04 as i/p - green button)
                    ;return to setup()
;-------------------------------------------------------------------------
btnLED:
    SBIC  PIND, 2             ;skip next statement if red button not pressed
    RJMP  redledON            ;jump to label redledON
    SBIC  PIND, 3             ;skip next statement if yellow button not pressed
    RJMP  yellowledON         ;jump to label yellowledON
    SBIC  PIND, 4             ;skip next statement if green button not pressed
    RJMP  greenledON          ;jump to label greenledON
    RJMP  btnLED              ;return to label btnLED
;-------------------------------------------------------------------------

redledON:
    LDI R21, 0b00010000       ;Set PB4 as high and the rest to low
    OUT PORTB, R21            ;Output R21 to PORTB
    RJMP btnLED               ;Return to label btnLED

yellowledON:
    LDI R21, 0b00001000       ;Set PB3 as high and the rest to low
    OUT PORTB, R21            ;Output R21 to PORTB
    RJMP btnLED               ;Return to label btnLED

greenledON:
    LDI R21, 0b00000100       ;Set PB2 as high and the rest to low
    OUT PORTB, R21            ;Output R21 to PORTB
    RJMP btnLED               ;Return to label btnLED
```



Contoh Kode Blink Bergantian




```
; Assembly Code
;---------------
#define __SFR_OFFSET 0x00
#include "avr/io.h"
;------------------------
.global main
;------------------------
main:
    LDI   R20, 0xFF
    OUT   DDRD, R20   ;set port D as o/p
agn:LDI   R20, 0x55
    OUT   PORTD, R20  ;o/p to port D the binary pattern 01010101
    RCALL myDelay     ;delay
    LDI   R20, 0xAA
    OUT   PORTD, R20  ;o/p to port D the binary pattern 10101010
    RCALL myDelay     ;delay
    RJMP  agn         ;repeat indefinitely
;-----------------------------------------------------------------
myDelay:              ;3-level nested loop subroutine
    LDI   R20, 255    ;outer loop counter
l1: LDI   R21, 255    ;mid loop counter
l2: LDI   R22, 41     ;inner loop counter to give 0.5s delay
l3: DEC   R22         ;decrement inner loop
```

```
    BRNE  l3          ;loop if not zero
    DEC   R21         ;decrement mid loop
    BRNE  l2          ;loop if not zero
    DEC   R20         ;decrement outer loop
    BRNE  l1          ;loop if not zero
    RET               ;return to caller
```