

---

# **DIKTAT PRAKTIKUM MIKROPROSESOR DAN MIKROPENGENDALI**



Disusun oleh:  
Arief Hendra Saptadi, S.T., M.Eng.

**PROGRAM STUDI S1 TEKNIK INFORMATIKA  
LABORATORIUM PEMROGRAMAN  
SEKOLAH TINGGI TEKNOLOGI TELEMATIKA TELKOM  
JL.DI.PANJAITAN 128 PURWOKERTO**

---



**LEMBAR PENGESAHAN  
MODUL PRAKTIKUM  
MIKROPROSESOR DAN MIKROPENGENDALI**

Purwokerto, 2017

Disusun oleh:

**Arief Hendra Saptadi, S.T., M.Eng.**

Mengesahkan,

Ketua Program Studi  
S1 Teknik Informatika

Ka.Bag Perpustakaan &  
Laboratorium

**Didi Supriyadi, S.T., M.Kom.**

**Risa Farrid Christianti, S.T., M.T.**

## DAFTAR ISI

Pendahuluan .....	1
Unit I. General Purpose Register .....	8
Unit II. Aritmatika Lanjutan dan Output Digital .....	16
Unit III. Kondisi Percabangan dan Input Digital .....	29

### **Tata Tertib Laboratorium**

1. Mahasiswa wajib mengenakan seragam resmi yang telah ditentukan ST3 Telkom dan memakai sepatu.
2. Mahasiswa wajib menjaga kebersihan ruang Laboratorium dan membuang sampah pada tempatnya.
3. Mahasiswa menggunakan fasilitas yang disediakan Laboratorium untuk aktivitas praktikum, workshop, pengujian alat tugas akhir dan segala kegiatan yang berhubungan laboratorium. Untuk kegiatan selain hal tersebut tidak diperbolehkan, tanpa seijin Ka.Ur. Laboratorium.
4. Selama berada di dalam Laboratorium, mahasiswa dilarang :
  - a. Membawa makanan atau minuman dan makan atau minum;
  - b. Mengambil atau membawa keluar alat/bahan yang disediakan Laboratorium, tanpa seijin *Officer* Laboran.
5. Mematuhi segala tata tertib dan aturan yang ditentukan oleh Ka.Ur. Laboratorium.

### **Tata Tertib Praktikum di Laboratorium**

#### **A. Sebelum Praktikum**

1. Praktikan wajib :
  - a. Menunjukkan Kartu Peserta Praktikum yang sudah diisi dan dilengkapi dengan foto berwarna terkini.
  - b. Menyediakan sendiri alat-alat tulis/gambar yang diperlukan.
  - c. Menguasai dasar teori dari unit modul praktikum yang akan dilakukan.
  - d. Membawa buku panduan praktikum, baik dalam bentuk *hardcopy* ataupun *softcopy*.
  - e. hadir tepat pada waktunya sesuai dengan jadwal yang telah ditentukan. Bila keterlambatan melebihi 10 menit maka yang bersangkutan tidak diperkenankan mengikuti praktikum dan baginya tidak diberikan praktikum susulan.
2. Praktikan akan *briefing* pada saat Pre-Test oleh Dosen Praktikum.
3. Praktikan diperbolehkan melakukan tukar-jadwal dengan praktikan lain setelah konfirmasi ke asisten praktikum dan mengisi formulir tukar-jadwal yang telah disediakan.

#### **B. Selama Praktikum**

1. Setiap unit modul sudah disediakan alat, tempat, dan bahan sendiri yang tidak boleh diubah, diganti, atau ditukar kecuali dengan sepengetahuan asisten.
2. Praktikan wajib membaca petunjuk langkah kerja dan mencatat hasil kerja praktikum yang tercantum dalam modul praktikum ataupun sesuai arahan asisten atau dosen pengampu.
3. Apabila menjumpai kesalahan, kerusakan, atau ketidaksesuaian dengan langkah kerja praktikum, praktikan harus segera melapor pada asisten.
4. Khusus untuk praktikum yang berhubungan dengan sumber arus atau tegangan, setelah selesai menyusun rangkaian sesuai langkah kerja, praktikan harus melapor kepada asisten, dan dilarang menghubungkan rangkaian dengan sumber tegangan atau arus tanpa seijin asisten.

5. Segala kerusakan yang terjadi karena kelalaian ataupun kesalahan praktikan akibat **tidak mengikuti langkah kerja praktikum** ditanggung oleh praktikan yang bersangkutan dan wajib untuk dilakukan penggantian paling lambat 1 (satu) minggu setelah terjadinya kerusakan.
6. Praktikan yang berhalangan praktikum, wajib memberitahukan kepada Dosen Praktikum maksimal 1 hari sebelum praktikum diadakan dengan menyertakan surat alasan tidak hadir saat praktikum dan bagi yang sakit menyertakan surat dokter (terkecuali bagi yang mendadak hari disaat praktikum yang bersangkutan sakit, ada pertimbangan tersendiri). Jika tidak, maka bagi yang bersangkutan diberikan praktikum susulan.
7. Praktikan tidak diperkenankan bersenda gurau dan atau meninggalkan ruangan praktikum tanpa seijin asisten atau dosen pengampu, serta bersikap tidak sopan terhadap para asisten atau dosen pengampu.
8. Praktikan diwajibkan mengembalikan alat-alat yang digunakan dan dilarang meninggalkan ruangan praktikum sebelum mendapat izin dari asisten atau pengampu praktikum.
9. Asisten praktikum berwenang memberikan tindakan terhadap Praktikan yang melanggar aturan, dengan sepengetahuan Dosen Praktikum.

#### **C. Setelah Praktikum**

1. Lembar data praktikum wajib mendapatkan persetujuan atau tanda tangan dari asisten, bila tidak maka data tersebut akan dinyatakan tidak sah.
2. Laporan praktikum dikumpulkan ke asisten sesuai dengan aturan yang telah ditetapkan sebelumnya.
3. Praktikan akan diberi pos-test oleh Dosen Praktikum dibantu oleh asisten praktikum.

#### **D. Ketentuan Lain**

1. Praktikum susulan diselenggarakan hanya untuk mahasiswa yang berhalangan hadir pada saat praktikum dikarenakan sakit, menikah, orang tua/wali atau saudara kandung meninggal, dan dispensasi mengikuti kegiatan dari kampus.
2. Praktikum susulan akan terselenggara, jika mahasiswa yang bersangkutan dapat menunjukkan surat keterangan resmi, seperti : Surat Keterangan Sakit dari dokter dan Surat Dispensasi dari bagian Akademik.
3. Penyelenggara praktikum susulan hanya diperbolehkan atas seijin Dosen Praktikum dan Ka.Ur. Laboratorium.

**MODUL PRAKTIKUM  
MIKROPROSESOR  
DAN MIKROPENGENDALI**

**PROGRAM STUDI  
S1 TEKNIK INFORMATIKA**



**Disusun Oleh:**

**Arief Hendra Saptadi, S.T., M.Eng.**

**LABORATORIUM TEKNIK  
ELEKTRONIKA/TEKNIK DIGITAL  
STT TELEMATIKA TELKOM PURWOKERTO  
JL. DI. PANJAITAN 128 PURWOKERTO**



**LABORATORIUM TEKNIK ELEKTRO & TEKNIK DIGITAL**  
**SEKOLAH TINGGI TEKNOLOGI TELEMATIKA TELKOM**  
**Jl. D. I. Panjaitan No. 128 Purwokerto**

---

## **LEMBAR PENGESAHAN**

Berdasarkan kajian substansi materi dan kebakuan tata tulis, maka modul praktikum:

### **MIKROPROSESOR DAN MIKROPENGENDALI**

yang disusun oleh:

**ARIEF HENDRA SAPTADI, S.T., M.Eng.**

dinyatakan:

**LAYAK**

untuk dipergunakan dalam penyelenggaraan praktikum pada Program Studi:

**S1 Teknik Informatika**

mulai:

**Tahun Akademik 2015 – 2016**

Mengetahui,

Purwokerto,

2015

Mengesahkan,

**Didi Supriyadi, S.T., M.Kom.**  
Ketua Program Studi

**Jaenal Arifin, S.T., M.Eng.**  
Kepala Lab. Teknik Elektro & Teknik Digital

Contoh cover utama :

# **LAPORAN PRAKTIKUM MIKROPROSESOR DAN MIKROPENGENDALI**

**PROGRAM STUDI  
S1 TEKNIK INFORMATIKA**

**Modul I :**

**Modul II :**

**Modul III :**



**DISUSUN OLEH :**  
<Nama Mahasiswa>  
<Nim>  
<Kelompok>

Dikumpulkan Tanggal : ... Mei 2015

Asisten Praktikum : <Nama Asisten 1>  
???

**LABORATORIUM  
TEKNIK ELEKTRONIKA DAN TEKNIK DIGITAL  
SEKOLAH TINGGI TEKNOLOGI TELEMATIKA TELKOM  
JL. D.I. PANJAITAN 128 PURWOKERTO**

**2015**



Contoh cover per modul :

# **LAPORAN PRAKTIKUM MIKROPROSESOR DAN MIKROPENGENDALI**

**PROGRAM STUDI  
S1 TEKNIK INFORMATIKA**

**Modul I :**



DISUSUN OLEH :  
<Nama Mahasiswa>  
<Nim>  
<Kelompok>

Tanggal Praktikum : ... Mei 2015

Asisten Praktikum : <Nama Asisten 1>  
... dst

**LABORATORIUM  
TEKNIK ELEKTRONIKA DAN TEKNIK DIGITAL  
SEKOLAH TINGGI TEKNOLOGI TELEMATIKA TELKOM  
JL. DI. PANJAITAN 128 PURWOKERTO**

**2015**

Contoh pembatas di setiap modul:

(kertas berwarna biru dengan isi logo STT Telematika TELKOM)



## **Sistematika Laporan Praktikum :**

### **I. Cover Utama**

### **II. Cover Modul 1**

- a. Tujuan Praktikum
- b. Alat dan Bahan
- c. Analisa
- d. Jawaban Pertanyaan
- e. Tugas Pemrograman

### **III. Cover Modul 2**

- a. Tujuan Praktikum
- b. ... dst...

### **IV. Cover Modul 3**

- a. Tujuan Praktikum
- b. ... dst...

### **V. Penutup**

- a. Kesimpulan (dari Modul 1 s.d. 4)
- b. Saran

## PENDAHULUAN

### I. Mikropengendali (*Microcontrollers*)

Mikropengendali adalah sebuah mikrokomputer (*microcomputer*) yang di dalamnya tidak hanya terdapat unit pemroses utama (*Central Processing Unit / CPU*), namun juga memasukkan berbagai perangkat (*peripheral*) dan memori ke dalam sebuah rangkaian terintegrasi tunggal berbentuk sebuah cip. Beberapa produsen mikropengendali dan produknya, antara lain adalah Intel (8031, 8051), Atmel (ATmega, ATtiny), PICAXE (PICAXE-08M2), Freescale/Motorola (S08, RS08, HC08), Texas Instruments (MSP430, C2000) dan Microchip (PIC).

Mikropengendali memiliki kesamaan dengan mikroprosesor (*microprocessor*) dalam hal bahwa keduanya memiliki CPU, namun terdapat berbagai perbedaan di antara keduanya, sebagai berikut:

**Tabel 1. Perbedaan antara Mikropengendali dengan Mikroprosesor**

Aspek	Mikropengendali	Mikroprosesor
Periferal dan Memori	Terintegrasi. Menjadi satu dengan CPU	Eksternal. Terpisah dari CPU.
Kemampuan Komputasi	Terbatas. Memiliki kecepatan pemrosesan yang rendah, berkisar pada puluhan hingga ratusan MHz	Canggih. Memiliki kecepatan pemrosesan yang tinggi, berkisar pada beberapa GHz
Aplikasi	Digunakan pada aplikasi sematan ( <i>embedded application</i> ) dengan sumber daya yang minim. Diterapkan untuk melakukan pekerjaan tertentu secara berulang-ulang.	Pada umumnya digunakan untuk aplikasi <i>desktop</i> atau <i>server</i> . Diterapkan untuk berbagai keperluan ( <i>general purpose</i> ).

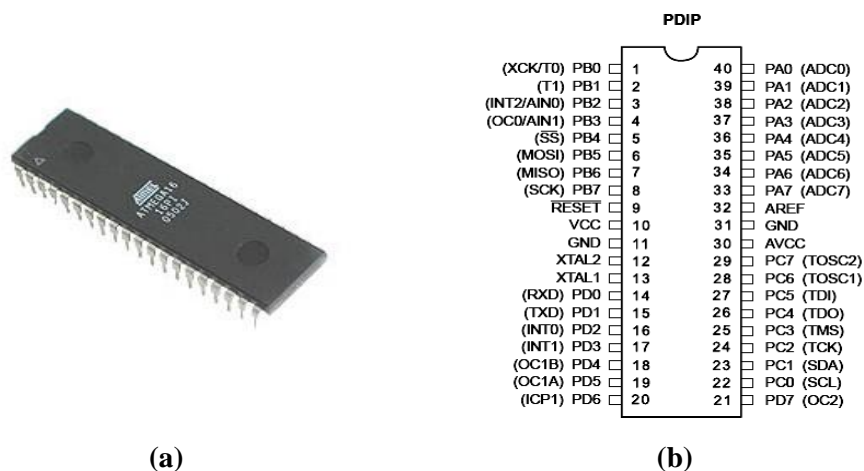
Berbagai perangkat dan memori yang umumnya terdapat di dalam mikropengendali antara lain adalah:

- Port masukan dan keluaran digital, yang terdiri dari beberapa lajur atau pin
- *Flash* yaitu memori untuk menyimpan program utama yang dijalankan oleh mikropengendali, bersifat dapat dihapus dan ditulis ulang.

- EEPROM (*Electrically Erasable Programmable Read Only Memory*) adalah memori yang digunakan untuk menyimpan data yang bersifat permanen.
- SRAM (*Static Random Access Memory*) merupakan memori yang menyimpan data pemrosesan secara sementara dan akan terhapus isinya saat catu daya dimatikan.
- Sistem interupsi (*Interrupts*) adalah sistem pensinyalan ke CPU yang menandakan bahwa sebuah kejadian (*event*) memerlukan perhatian khusus.
- *Timer/Counter* yaitu bagian dari mikropengendali yang bertugas untuk melakukan pewaktuan atau pencacahan. *Timer/Counter* juga digunakan untuk mengatur pulsa aktif selama satu periode (*duty cycle*), yang diterapkan dalam teknik modulasi PWM (*Pulse Width Modulation*).
- ADC (*Analog to Digital Converter*) adalah perangkat mikropengendali yang bertugas untuk mengubah masukan analog (lazimnya dari perangkat sensor) menjadi keluaran digital yang akan diproses oleh CPU.
- USART (*Universal Synchronous Asynchronous Receiver Transmitter*) merupakan bagian pengendali komunikasi serial.

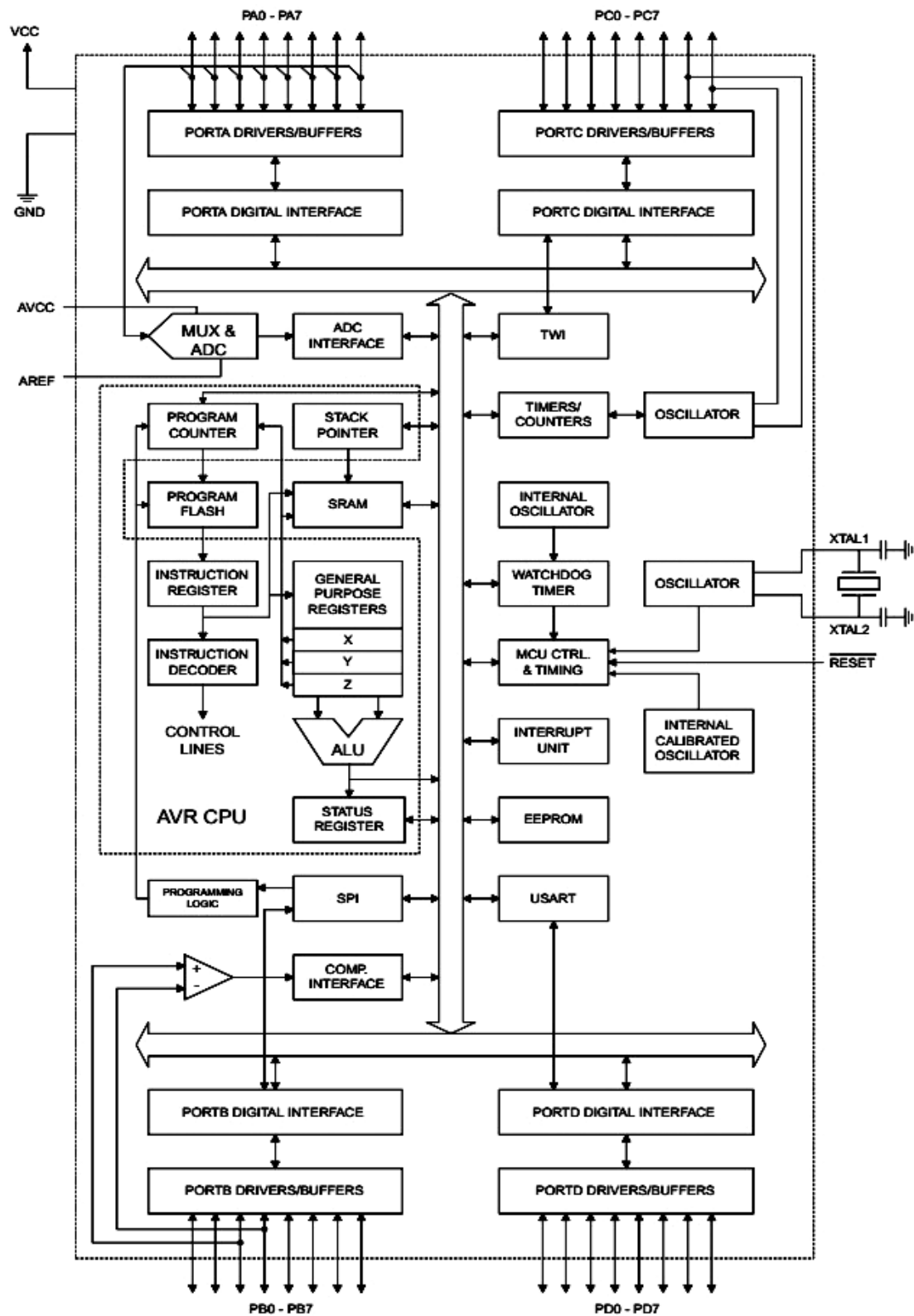
## II. Arsitektur Mikropengendali ATmega16A

ATmega16A adalah mikropengendali 8 bit keluaran ATMEL, yang termasuk famili megaAVR. ATmega16A merupakan versi optimasi dari ATmega16 dengan konsumsi arus yang lebih rendah. Mikropengendali ini memiliki 40 buah pin dan jenis kemasan yang digunakan untuk praktikum ini adalah PDIP-P dengan bentuk fisik seperti dalam gambar 1.(a).



**Gambar 1. ATmega16A (a) Bentuk Fisik (b) Konfigurasi Pin**

Bagian-bagian di dalam mikropengendali ATmega16A adalah sebagaimana diperlihatkan di dalam diagram blok berikut ini.

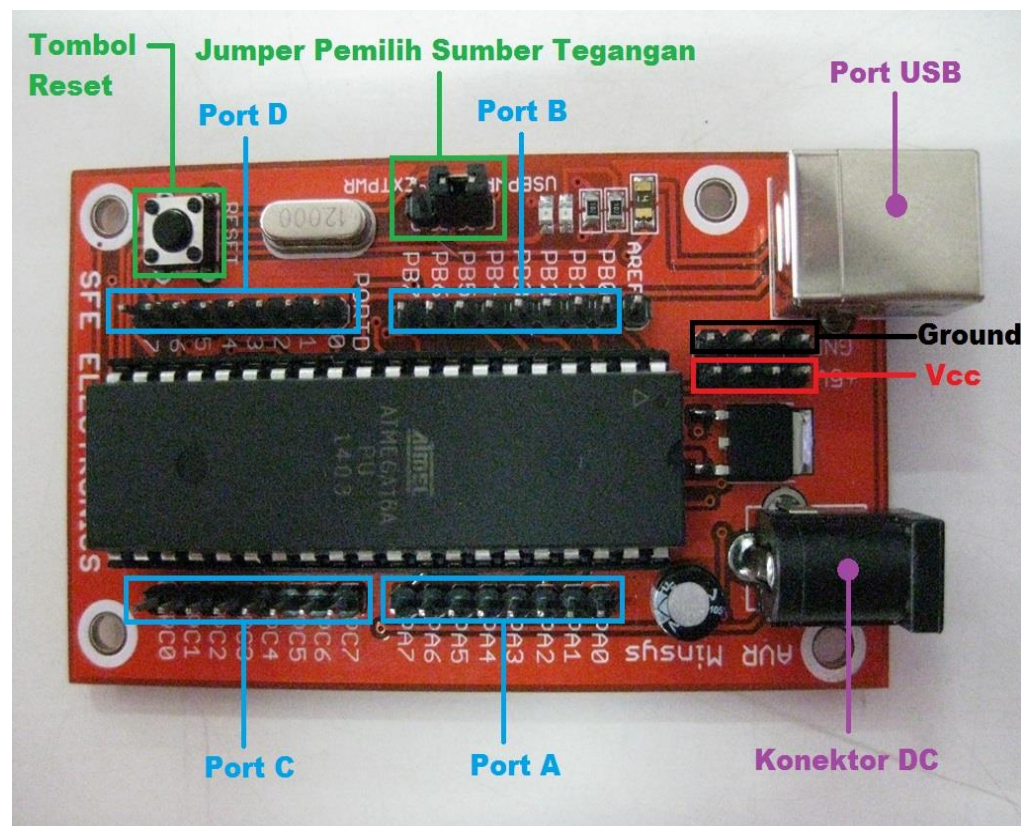


Gambar 2. Diagram Blok ATmega16A

Fitur-fitur dari ATmega16A seperti yang terlihat dalam diagram blok tersebut antara lain meliputi:

- Memori program Flash 16 KB
- EEPROM 512 Byte
- SRAM 1 KB
- Dua *Timer/Counter* 8 bit dan satu *Timer/Counter* 16 bit
- 8 kanal ADC (untuk jenis kemasan PDI-P) dengan akurasi 8 bit/10 bit
- Komunikasi Serial USART yang dapat diprogram
- 32 lajur masukan/keluaran digital yang dapat diprogram
- Sumber interupsi internal maupun eksternal
- Beroperasi pada tegangan 2,7 – 5,5 Volt dengan frekuensi 0 – 16 MHz

Pada praktikum ini digunakan sistem minimum ATmega16A dengan bagian-bagian seperti terlihat dalam gambar 2. Rangkaian ini menggunakan frekuensi detak (*clock*) sebesar 12 MHz. Tegangan sumber yang digunakan bisa berasal dari port USB yang dihubungkan ke komputer atau melalui konektor DC yang disambungkan ke adaptor. Pemilihan sumber tegangan ini dilakukan dari sebuah *jumper*.



Gambar 3. Papan Rangkaian Sistem Minimum ATmega16A

Ada pun port dan fungsi yang terdapat untuk masing-masing pin dalam mikropengendali ATmega16A adalah seperti dalam Tabel 2 berikut.

**Tabel 2. Port dan Fungsi Pin**

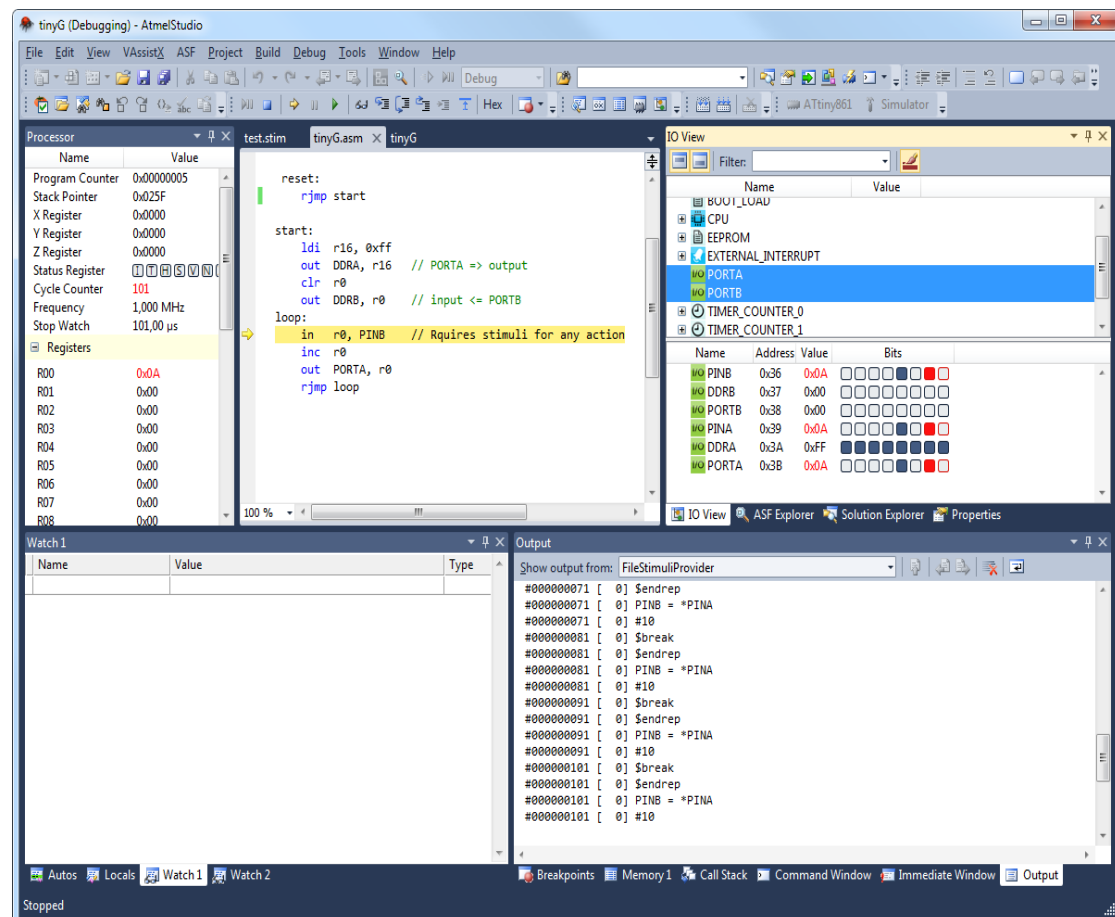
Pin No.	Port	Fungsi Utama	Fungsi Alternatif
1 – 8	PB0..7	Masukan/ Keluaran	PB0 memiliki fungsi alternatif sebagai masukan T0 atau pembangkit detak (clock) eksternal (XCK) untuk komunikasi serial (USART). PB1 bisa juga digunakan untuk masukan T1. PB2 dapat berfungsi sebagai masukan analog AIN0 atau sumber interupsi INT2. PB3 dapat berfungsi untuk masukan analog AIN1 atau masukan pewaktu OC0. Sedangkan PB4 sampai PB7 memiliki fungsi-fungsi untuk komunikasi menggunakan protokol SPI, masing-masing sebagai SS ( <i>Slave Select</i> ), MOSI ( <i>Master Output Slave Input</i> ), MISO ( <i>Master Input Slave Output</i> ) dan SCK ( <i>Slave Clock</i> ).
9	RESET	Masukan	Memberikan masukan kepada mikropengendali untuk mengulang instruksi dari awal
10	VCC	Tegangan Sumber	
11, 31	GND	Pentanahan	
12	XTAL2	Masukan	Masukan untuk rangkaian osilator
13	XTAL1		
14 – 20	PD0..6	Masukan/Keluaran	PD0 dan PD1 masing-masing dapat berfungsi sebagai penerima (RXD) dan pengirim (TXD) pada komunikasi serial. PD2 dan PD3 masing-masing dapat pula difungsikan sebagai masukan interupsi eksternal (INT0 dan INT1). PD4 dan PD5 dapat berfungsi untuk pembanding keluaran Timer/Counter 1 pin A dan B (OC1A dan OC1B). PD6 juga berfungsi untuk masukan pewaktu (ICP1).
21	PD7	Masukan/Keluaran	PD7 juga dapat digunakan untuk keluaran pewaktu (OC2).
22 – 29	PC0..7	Masukan/Keluaran	PC0 dan PC1 dapat berfungsi untuk komunikasi data menggunakan protokol I2C ( <i>Inter-Integrated Circuit</i> ) sebagai SCL ( <i>Two-Wire Serial Bus Clock Line</i> ) dan SDA ( <i>Two-Wire Serial Bus Data Input/Output Line</i> ). PC2 sampai dengan PC5 dapat berfungsi untuk protokol pemrograman JTAG yaitu masing-masing sebagai pemicu detak (TCK), pemilih mode (TMS), data keluaran (TDO) dan data masukan (TDI). PC6 dan PC7 masing-masing dapat berfungsi untuk pemicu detak bagi Timer/Counter 1 (TOSC1) dan Timer/Counter 2 (TOSC2).
30	AVCC	Tegangan Masukan untuk ADC	
32	AREF	Tegangan Referensi untuk ADC	
33 – 40	PA7..0	Masukan/Keluaran	PA0 hingga PA7 dapat juga berfungsi sebagai masukan analog untuk proses ADC



### III. ATMEL Studio

ATMEL Studio adalah lingkungan pengembangan yang terpadu (*Integrated Development Environment*) untuk seluruh mikropengendali yang dihasilkan oleh ATMEL. Dahulu bernama AVR Studio, saat ini ATMEL Studio mendukung pengembangan aplikasi untuk seri AVR (meliputi famili untuk aplikasi otomotif, aplikasi baterai dan catu daya, ATtiny, ATmega, ATXmega dan UC3), seri ARM dan seri arsitektur 8051.

Fitur penting dari ATMEL Studio adalah dukungannya untuk membuat aplikasi melalui bahasa C/C++ dan Assembly. Jenis *compiler* standard yang didukung adalah GCC dan ini banyak digunakan pada berbagai proyek *open source*. Selain memiliki kemampuan untuk mengompilasi kode sumber (*source code*) menjadi berkas HEX untuk diunduh ke mikropengendali, ATMEL Studio juga memiliki fitur simulasi. Pada proses simulasi tersebut, cara kerja dari program dapat diketahui demikian pula dengan berbagai isi dari variabel, register maupun memori (baik SRAM maupun EEPROM). Fitur ini sangat membantu untuk proses pelacakan kesalahan (*debugging*).



Gambar 4. Tampilan dari ATMEL Studio

#### IV. eXtreme Burner - AVR

eXtremeBurner - AVR adalah aplikasi yang digunakan untuk mengunduh program dalam bentuk berkas HEX dari hasil kompilasi menuju memori pada mikropengendali yaitu memori program atau Flash dan memori data atau EEPROM. Aplikasi ini terutama digunakan untuk mengunduh hasil kompilasi dari ATMEL Studio menuju mikropengendali ATmega16A melalui protokol USBASP.

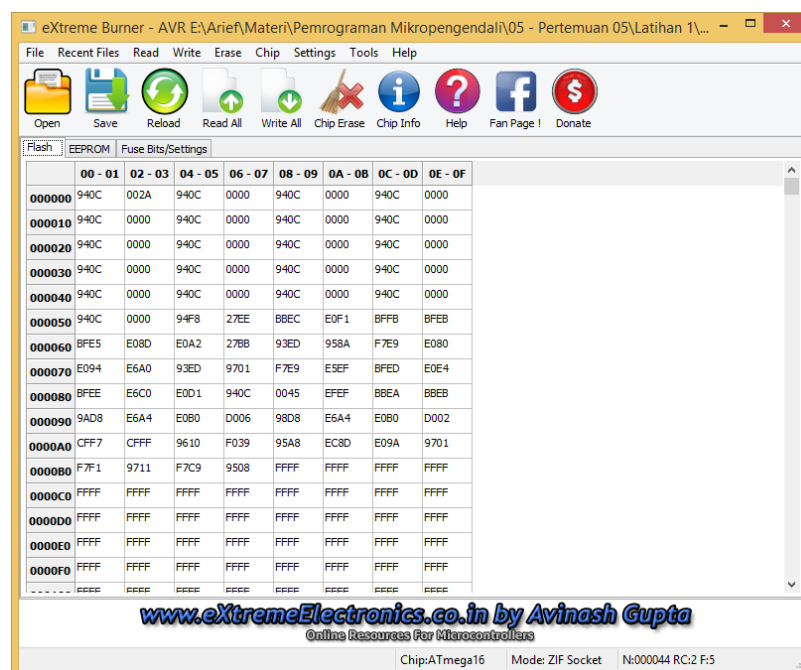
Fitur-fitur yang terdapat di dalam aplikasi ini antara lain:

- Membaca peta memori dari Flash dan EEPROM dengan isi data di dalamnya
- Membaca dan mengatur fuse bit pada mikropengendali
- Menghapus isi seluruh memori dari mikropengendali
- Mengamati data pada port Serial melalui Serial Terminal

Jenis-jenis mikropengendali yang didukung oleh aplikasi ini adalah seperti tertera di dalam Tabel 3 berikut.

**Tabel 3. Seri Mikropengendali yang Didukung eXtremeBurner - AVR**

Famili	
ATTiny	ATmega
ATTiny13A, ATTiny24, ATTiny44, ATTiny84, ATTiny2313	ATmega48, ATmega88, ATmega168, ATmega169P, ATmega8515, ATmega8535, ATmega8, ATmega16, ATmega162, ATmega164PA, ATmega324PA, ATmega32, ATmega64A, ATmega128, ATmega640, ATmega2560, AT90USB1268



**Gambar 5. Tampilan eXtreme Burner - AVR**

## UNIT I

### General Purpose Register

#### I. TUJUAN PRAKTIKUM

1. Mahasiswa mampu mengoperasikan ATMEL Studio untuk kompilasi dan simulasi kode program Assembly
2. Mahasiswa memahami struktur bahasa pemrograman Assembly
3. Mahasiswa mampu membuat program untuk mengatur nilai dalam *General Purpose Register* (GPR)
4. Mahasiswa mampu membuat program untuk operasi aritmatika sederhana

#### II. ALAT DAN BAHAN

1. 1 unit laptop (dari peserta praktikum) atau 1 unit PC (dari Lab. Komputer).
2. Perangkat lunak ATMEL Studio 6
3. Lembar Pencatatan

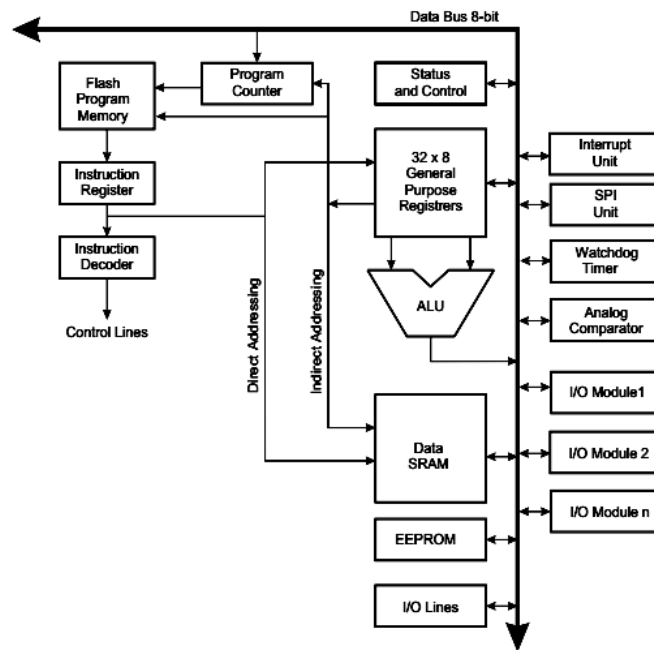
#### III. DASAR TEORI

##### A. CPU pada AVR

Fungsi utama dari CPU adalah untuk memastikan eksekusi program secara benar, yaitu dengan mengakses memori, melakukan penghitungan, mengendalikan perangkat dan menangani interupsi. Bagian utama dari CPU adalah ALU atau *Arithmetic and Logical Unit*. Bagian ini mendukung operasi aritmatika dan logika yang terjadi antar register, antara konstanta dengan register atau di dalam register tunggal. Setiap kali terjadi operasi aritmatika, isi dari Status Register diperbarui untuk memberikan informasi tentang hasil dari operasi tersebut.

Interaksi antara satu bagian dengan lainnya seperti pada Gambar 6 tersebut terjadi lewat pengalamatan (*addressing*). Ada dua jenis pengalamatan yaitu secara langsung (*direct addressing*) maupun tidak langsung (*indirect addressing*).

Mikropengendali ATMEL AVR menggunakan arsitektur Harvard, yaitu dimana terdapat pemisahan antara memori dan bus untuk program dan data. Ketika sebuah instruksi sedang dijalankan, instruksi berikutnya diantrikan dari memori program yang terletak di dalam Flash. Demikian seterusnya, sehingga untuk setiap satu siklus waktu bisa terjadi satu kali eksekusi.

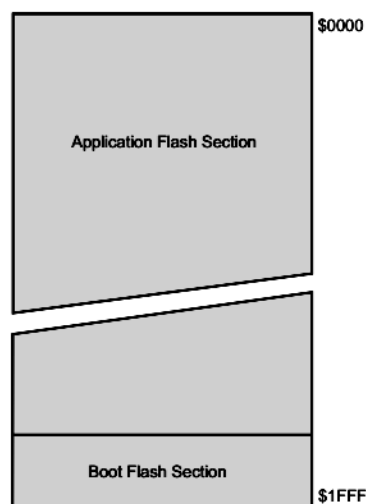


Gambar 6. Diagram Blok dari Arsitektur Mikropengendali AVR

## B. Memori dalam AVR

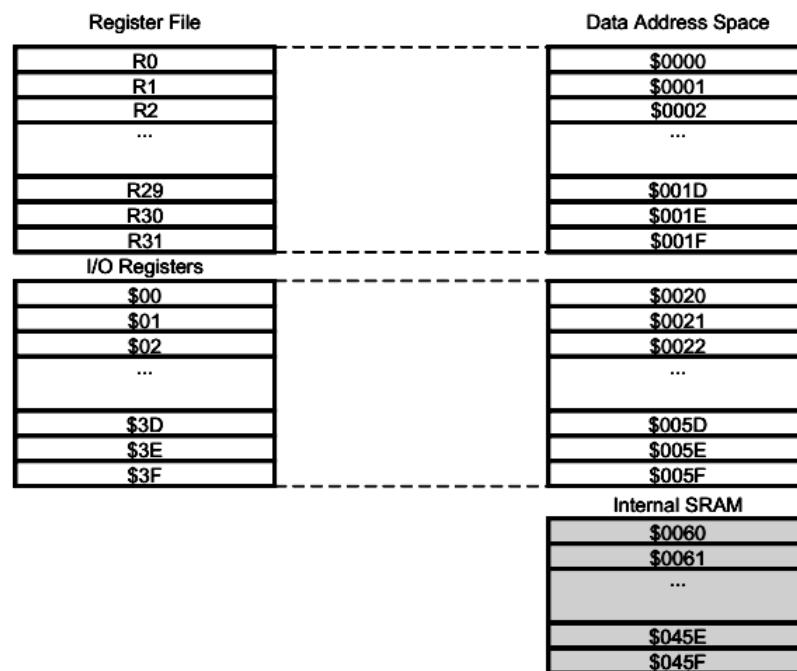
AVR memiliki tiga jenis memori, yaitu memori program berbentuk Flash (*Flash Program Memory*), memori sementara berwujud SRAM dan memori data berupa EEPROM. Ketiga memori tersebut dapat dibaca dan ditulis setiap saat, namun hanya SRAM yang akan terhapus isinya bila catu daya dimatikan (*power down*).

ATmega16A memiliki memori program sebesar 16KB. Karena panjang sebuah instruksi untuk AVR tersebut adalah 16 bit atau 32 bit, maka ukuran dari Flash diatur sebesar 8KB x 16 bit. Kapasitas memori program terbagi menjadi dua bagian, yaitu *Boot Program* dan *Application Program*.



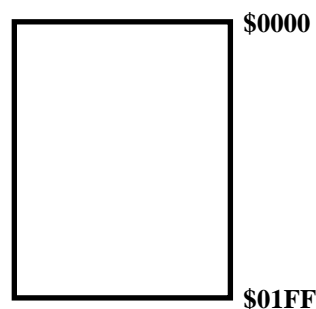
Gambar 7. Peta Memori Program pada Mikropengendali AVR

Memori data SRAM memiliki kapasitas sebesar 1120 Byte. Memori ini terbagi tiga yaitu untuk General Purpose Register (GPR) 32 Byte, I/O Register 64 Byte dan memori internal sebesar 1024 Byte. GPR menempati alamat memori terbawah yaitu dari \$0000 hingga \$001F, yang masing-masing dialokasikan untuk R0 sampai dengan R31 (32 buah register). I/O Registers berupa sejumlah register untuk mengatur operasi input dan output pada AVR menempati alamat \$0020 sampai dengan \$005F. Ada pun memori internal SRAM berawal dari alamat \$0060 hingga \$045F (64 buah register).



**Gambar 8. Peta SRAM pada Mikropengendali AVR**

EEPROM pada AVR memiliki kapasitas sebesar 512 byte dengan alamat memori yang terpisah yaitu dari \$0000 hingga \$01FF. Operasi yang terjadi antara CPU dengan EEPROM diatur melalui tiga buah register, yaitu EEPROM Address Register, EEPROM Data Register dan EEPROM Control Register.



**Gambar 9. Peta EEPROM pada Mikropengendali AVR**

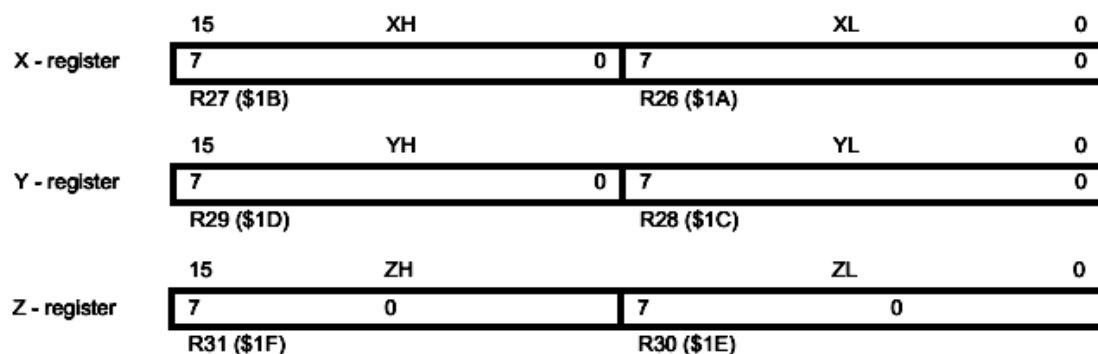
### C. General Purpose Registers (GPR)

GPR terdiri dari 32 buah register 8 bit (R0 – R31) yang berfungsi sebagai akumulator dalam operasi aritmatika maupun logika. R16 – R31 dapat dialamati secara langsung (*immediate addressing*). R26 hingga R31 memiliki fungsi khusus sebagai *pointer* pengalamatan 16 bit untuk *Data Space*, yaitu register X, Y dan Z.

	7	0	Addr.	
General Purpose Working Registers	R0		\$00	
	R1		\$01	
	R2		\$02	
	...			
	R13		\$0D	
	R14		\$0E	
	R15		\$0F	
	R16		\$10	
	R17		\$11	
	...			
	R26		\$1A	X-register Low Byte
	R27		\$1B	X-register High Byte
	R28		\$1C	Y-register Low Byte
	R29		\$1D	Y-register High Byte
	R30		\$1E	Z-register Low Byte
	R31		\$1F	Z-register High Byte

Gambar 10. General Purpose Registers (GPR)

Register X dibentuk oleh R26 dan R27 yang masing-masing mengisi 8 bit bawah (*Low Byte*) dan 8 bit atas (*High Byte*). Demikian pula untuk R28 dan R29 untuk register Y dan R30 dan R31 untuk register Z, seperti pada gambar berikut.



Gambar 11. Register X, Y dan Z

### D. Instruksi Dasar untuk GPR

#### 1. LDI

LDI atau *Load Immediate* berfungsi untuk menyalin nilai data 8 bit ke dalam GPR. Sintaks untuk LDI adalah sebagai berikut:

**LDI Rd, K**

Rd adalah register dari R26 hingga R31. K adalah suatu nilai 8 bit yaitu 0b00000000 – 0b11111111 dalam biner, 0 – 255 dalam desimal, 00 – 0377 untuk oktal atau 0x00 – 0xFF untuk heksadesimal.

Contoh:     LDI R27, 0b10110110     ; Memasukkan biner 10110110  
              LDI R27, 182             ; Memasukkan desimal 182  
              LDI R27, 0xB6           ; Memasukkan heksadesimal B6  
              LDI R27, 0266           ; Memasukkan oktal 266

Beberapa ketentuan untuk penggunaan perintah LDI:

1. LDI tidak dapat memasukkan nilai secara langsung ke register R0 hingga R15.

Contoh:     LDI R3, 0xAB             ; Salah! Harus di R26 s.d. R31

2. Jika bilangan yang dimasukkan hanya terdiri dari satu digit, maka diasumsikan di depan bilangan tersebut terdapat angka nol.

Contoh:     LDI R18, 0xC             ; Sama saja R18 = 0x0C

3. Perintah LDI untuk memasukkan nilai lebih besar daripada 255 (desimal), 0377 (oktal), 0b11111111 (biner) atau 0xFF (heksadesimal) akan memicu kesalahan.

Contoh:     LDI R25, 0401             ; Salah! Nilai oktal maksimal  
  ; adalah 0377

## 2. ADD

ADD adalah perintah yang digunakan untuk melakukan penjumlahan nilai pada dua buah register dengan hasil disimpan pada register yang dituliskan pertama. Sintaks untuk ADD adalah sebagai berikut:

**ADD Rd, Rr**

Rd dan Rr adalah R16 hingga R31. Hasil penjumlahan disimpan di Rd.

Contoh:     ADD R19, R18             ; Nilai R18 & R19 dijumlahkan  
  ; Hasilnya disimpan di R19

ADD tidak dapat digunakan untuk menjumlahkan suatu nilai secara langsung ke sebuah register.

Contoh:     ADD R20, 0b10100011     ; Tidak bisa dijumlah langsung  
  Penjumlahan nilai antara dua buah register hendaknya melibatkan perintah LDI untuk memasukkan nilai ke masing-masing register.

Contoh:     LDI R18, 0x1C             ; R18 = 0x1C  
              LDI R19, 0x92             ; R19 = 0x92  
              ADD R18, R19             ; R18 = 0x1C + 0x92 = 0xAE

#### IV. PROSEDUR PRAKTIKUM

##### A. Percobaan 1-1

1. Buka ATMEL Studio. Buat sebuah project baru dan namai dengan Percobaan 1-1. Ketik listing berikut ini dan lakukan kompilasi. Catat langkah-langkah yang anda lakukan di dalam lembar yang disediakan.

```
.nolist
.include "m16def.inc"
.list
.def register = R16
    LDI register, 0b11111111
    OUT DDRB, register
label1:
    LDI register, 0b01010101
    OUT PORTB, register
    LDI register, 0b10101010
    OUT PORTB, register
    RJMP label1
```

2. Untuk listing di atas, jalankan simulasi. Catat langkah-langkah melakukan simulasi ke dalam lembar tersebut.

##### B. Percobaan 1-2

1. Buka sebuah project baru, namakan dengan Percobaan 1-2. Ketik listing berikut ini:

```
;Load Immediate
.nolist
.include "m16def.inc"
.def register = R17
.list
.org 0x00
    LDI register, 0xA2 ; Desimal = 162, Oktal = 242
```

2. Lakukan kompilasi dan jalankan simulasi.
3. Ganti nilai 0xA2 tersebut dengan 162. Ulangi simulasi.
4. Ganti lagi menjadi 0242. Ulangi simulasi.
5. Apa kesimpulan anda?
6. Apa fungsi perintah .def di atas?
7. Berapa nilai akhir R17?
8. Ganti R17 menjadi R3. Jalankan kompilasi. Apa yang terjadi? Mengapa demikian?
9. Catat jawaban pertanyaan di nomor 5, 6, 7 dan 8 ke dalam lembar praktikum.



**C. Percobaan 1-3**

1. Buka sebuah project baru, namakan dengan Percobaan 1-3. Ketik listing berikut ini:

```
.nolist
.include "m16def.inc"
.list
.org 0x00
    LDI R16, 0x14
    LDI R17, 0x23
    ADD R16, R17
```

2. Lakukan kompilasi dan jalankan simulasi. Pada saat simulasi berjalan, isilah nilai dari tiap register untuk per baris instruksi seperti dalam tabel berikut:

**Tabel 4. Isi Register untuk Percobaan 1-3**

Instruksi	Isi Register	
	R16	R17
LDI R16, 0x14		
LDI R17, 0x23		
ADD R16, R17		

**D. Percobaan 1-4**

1. Buka sebuah project baru, namakan dengan Percobaan 1-4. Ketik listing berikut ini:

```
.nolist
.include "m16def.inc"
.list
.org 0x00
    LDI R18, 0xEA
    LDI R19, 0xF1
    MOV R17, R19
    MOV R1, R18
```

2. Lakukan kompilasi dan jalankan simulasi. Pada saat simulasi berjalan, isilah nilai dari tiap register untuk per baris instruksi seperti dalam Tabel 5 pada halaman berikutnya.

**Tabel 5. Isi Register untuk Percobaan 1-4**

Instruksi	Isi Register			
	R1	R17	R18	R19
LDI R18, 0xEA				
LDI R19, 0xF1				
MOV R17, R19				
MOV R1, R18				

## V. TUGAS PEMROGRAMAN

Buatlah sebuah program di dalam Assembly yang melakukan hal-hal berikut ini:

1. Menamai register **R18, R19** dan **R20** menjadi **nilai1, nilai2** dan **hasil**.
2. Memasukkan nilai **0x15** ke **nilai1** dan **0x97** ke **nilai2**
3. Menyalin isi **nilai1** ke **hasil** dan menjumlahkan antara **hasil** dengan **nilai2**

Buat tabel isi register untuk instruksi 2 dan 3 tersebut seperti tabel di dalam percobaan 1-3 dan 1-4. Gambarkan juga diagram alirnya!

## UNIT II

### Aritmatika Lanjutan dan Output Digital

#### I. TUJUAN PRAKTIKUM

1. Mahasiswa mampu membuat program untuk operasi aritmatika lanjutan
2. Mahasiswa mampu membuat program untuk operasi output digital
3. Mahasiswa mampu merakit rangkaian untuk operasi output digital
4. Mahasiswa mampu mengunduh program hasil kompilasi ke mikropengendali

#### II. ALAT DAN BAHAN

1. 1 unit laptop (dari peserta praktikum) atau 1 unit PC (dari Lab. Komputer).
2. Perangkat lunak ATMEL Studio 6 dan ExtremeBurner AVR 1.4.2
3. 1 set minimum system ATmega16
4. 1 set kabel male-to-male
5. 1 set kabel male-to-female
6. 8 buah LED 5mm
7. 8 buah resistor 220  $\Omega$
8. Lembar Pencatatan

#### III. DASAR TEORI

##### A. Rangkaian Output

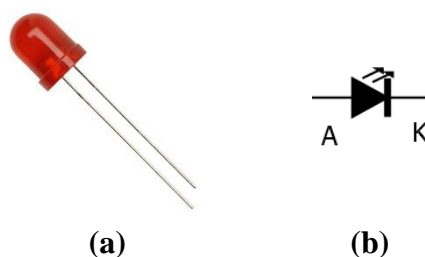
Mikropengendali ATmega16A memiliki 32 pin, terbagi ke dalam 4 port yang dapat dimanfaatkan sebagai lajur masukan atau keluaran. Port dan pin tersebut adalah sebagai berikut:

**Tabel 6. Port dan Pin pada ATmega16A**

Port	Pin	Jumlah Bit
A	0..7 (PA0 – PA7)	8
B	0..7 (PB0 – PB7)	8
C	0..7 (PC0 – PC7)	8
D	0..7 (PD0 – PD7)	8

Jika suatu pin digunakan sebagai output, maka lazimnya LED digunakan sebagai indikator yang menunjukkan bahwa pin itu diberikan logika nol atau satu.

LED atau *Light Emitting Diode* adalah komponen elektronika yang akan mengeluarkan cahaya saat dialiri oleh arus listrik. LED memiliki dua kaki, yaitu kaki positif atau anoda (A) dan kaki negatif atau katoda (K). Anoda ditandai oleh kaki yang lebih panjang, sementara Katoda berupa kaki yang lebih pendek. Bentuk fisik dan simbol komponen dari LED tersebut adalah seperti terlihat dalam Gambar 12.

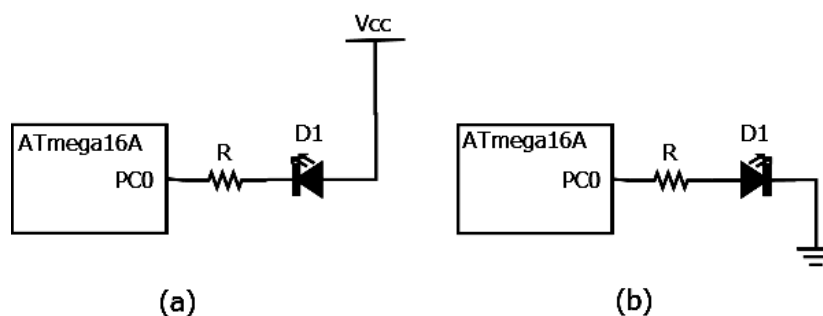


**Gambar 12. *Light Emitting Diode* (a) Bentuk Fisik (b) Simbol Komponen**

Dalam penggunaannya, LED harus dipasang seri dengan sebuah resistor untuk membatasi arus yang mengalirinya. Hal ini karena LED memiliki nilai tahanan dalam yang sangat kecil, sehingga jika diberikan tegangan yang tidak terlalu besar nilainya sekalipun, itu sudah cukup untuk memberikan arus yang besar dan dapat merusakkannya. Sebuah resistor seri ( $R$ )  $220\ \Omega$  untuk setiap LED sudah mencukupi.

Berdasarkan orientasi pemasangannya terhadap tegangan sumber ( $V_{CC}$ ) atau *ground*, terdapat dua jenis rangkaian output yaitu *Active Low* dan *Active High* seperti pada Gambar 13. Pada contoh tersebut, pin PC0 dari ATmega16A digunakan untuk memberikan keluaran yang akan mengatur menyala atau matinya LED.

Pada rangkaian *Active Low*, jika PC0 diberikan logika nol, maka terdapat arus yang mengalir dari  $V_{CC}$  menuju PC0 sehingga LED pada D1 menyala. Saat PC0 diberikan logika satu, maka pada pin tersebut terdapat tegangan 5 Volt yang sama dengan tegangan pada  $V_{CC}$ . Sehingga terjadi selisih tegangan antara keduanya dan ini menyebabkan tidak ada arus yang mengalir di D1. LED menjadi mati.



**Gambar 13. Rangkaian Output (a) Active Low (b) Active High**

Pada rangkaian *Active High*, jika PC0 diberikan nilai logika 1, maka terdapat tegangan pada pin tersebut dan ada arus yang mengalir di D1. LED menyala. Saat PC0 diberikan nilai logika 0, maka sama sekali tidak terdapat arus yang mengalir di D1. LED menjadi mati.

## B. Register-Register Pengatur Fungsi Output

Pengaturan pin pada mikropengendali sebagai keluaran melibatkan dua buah register seperti berikut ini:

### 1. DDRx

DDRx atau *Data Direction Register* adalah register 8 bit yang berfungsi untuk mengatur apakah suatu pin bertugas sebagai masukan atau keluaran. Huruf x di belakang DDR tersebut menunjukkan nama dari port. Pada ATmega16A terdapat empat port untuk input/output, yaitu A, B, C dan D sehingga terdapat empat buah register DDRx yaitu DDRA, DDRB, DDRC dan DDRD.

Jika suatu pin difungsikan sebagai output maka pin tersebut diberikan nilai satu, namun jika sebagai input maka diberikan nilai nol. Di dalam port yang sama dapat dilakukan pengaturan untuk pin mana saja yang berfungsi sebagai input dan output. Dalam bahasa Assembly, pengaturan pin ini melibatkan perintah LDI dan OUT. Perintah LDI tersebut untuk memasukkan nilai 8 bit yang menunjukkan pengaturan pin ke GPR. Isi dari GPR tersebut selanjutnya disalin ke register DDRx melalui perintah OUT.

Contoh:

Pada mikropengendali ATmega16A, empat pin terbawah dari port D akan digunakan sebagai input dan empat pin teratas sebagai output. Dengan demikian maka pada register DDRD untuk pin PD0 sampai PD3 diberikan nilai 0 sedangkan pin PD4 hingga PD7 diberikan nilai 1, seperti pada gambar berikut ini.

**Register DDRD:**

PD7	PD6	PD5	PD4	PD3	PD2	PD1	PD0
1	1	1	1	0	0	0	0

0xF0

**Gambar 14. Pengaturan Register DDRD**

Sesuai Gambar 14, register DDRD diberikan nilai 0xF0. Pemberian nilai tersebut dalam bahasa Assembly adalah seperti di bawah ini:

```
LDI R17, 0xF0          ; Memasukkan 0xF0 ke R17
OUT DDRD, R17          ; Mengatur DDRD sesuai nilai
                        ; pada R17 yaitu 0xF0
```

## 2. PORTx

PORTx adalah register 8 bit yang berfungsi untuk memberikan nilai logika nol atau satu ke suatu pin. Jika suatu pin bertindak sebagai keluaran, maka pin tersebut dapat diatur untuk mengeluarkan logika nol atau satu via register PORTx.

Sama halnya dengan DDRx, untuk mikropengendali ATmega16A terdapat empat buah register PORTx yaitu PORTA, PORTB, PORTC dan PORTD. Pengaturan nilai PORTx menggunakan bahasa Assembly juga melibatkan perintah LDI dan OUT. Sebelum memberikan nilai ke PORTx, nilai dari register DDRx untuk port tersebut harus diatur terlebih dahulu.

Contoh:

Bila port A seluruhnya digunakan sebagai output dan semua pin mengeluarkan logika satu kecuali PA0, maka pengaturan untuk register-registernya adalah seperti dalam gambar berikut.

### Register DDRA:

PA7	PA6	PA5	PA4	PA3	PA2	PA1	PA0
1	1	1	1	1	1	1	1

0xFF

### Register PORTA:

PA7	PA6	PA5	PA4	PA3	PA2	PA1	PA0
1	1	1	1	1	1	1	0

0xFE

**Gambar 15. Pengaturan Register DDRA dan PORTA**

Sesuai Gambar 15, register DDRA dan PORTA masing-masing diberikan nilai 0xFF dan 0xFE. Pemberian nilai-nilai untuk kedua register tersebut dalam bahasa Assembly adalah seperti di bawah ini:

```
Contoh:  LDI R19, 0xFF          ; Memasukkan 0xFF ke R19
          OUT DDRA, R19         ; Mengatur DDRD sesuai nilai
                                   ; pada R19 yaitu 0xFF
          LDI R19, 0xFE         ; Memasukkan 0xFE ke R19
          OUT PORTA, R19        ; PORTA diberi nilai 0xFE
```

### C. Instruksi-Instruksi Perulangan, Aritmatika dan Output

#### 1. MOV

Perintah MOV atau kependekan dari Move ini digunakan untuk menyalin nilai antar *General Purpose Register*. Sintaks untuk MOV adalah seperti berikut ini.

**MOV Rd, Rr**

Ada pun Rd dan Rr tersebut adalah semua register yang termasuk *General Purpose Register* (GPR).

```
Contoh:  MOV R20, R21          ; Menyalin isi R21 ke R20
```

Berbeda dengan LDI yang hanya dapat mengakses R16 hingga R31, perintah MOV ini dapat mengakses seluruh GPR. Oleh karenanya hal ini dapat dimanfaatkan untuk memberikan nilai ke R0 sampai dengan R15 secara tidak langsung dengan bantuan LDI seperti contoh di bawah ini.

```
Contoh:  LDI R16, 0x13          ; Memberi nilai 0x13 ke R16
          MOV R2, R16           ; Menyalin nilai 0x13 ke R2
```

Perintah MOV tidak dapat digunakan untuk mengisi nilai secara langsung ke sebuah GPR. Gunakan LDI untuk keperluan tersebut.

```
Contoh:  MOV R17, 0x45          ; Salah! Gunakan LDI
```

#### 2. INC

Perintah INC atau Increment berfungsi untuk menaikkan nilai dari suatu GPR sebanyak 1. Sintaks untuk perintah ini adalah seperti berikut.

**INC Rd**

Ada pun Rd adalah *General Purpose Register* (GPR) dari R0 sampai dengan R31. Setelah perintah INC dijalankan maka nilai Rd menjadi  $Rd = Rd + 1$ .

Contoh:     LDI R30, 0x8A                     ; R30 = 0x8A  
              INC R30                         ; R30 menjadi 0x8B

### 3. DEC

Perintah DEC atau Decrement berfungsi untuk menurunkan nilai dari suatu GPR sebanyak 1. Sintaks untuk perintah ini adalah seperti berikut.

DEC Rd

Sedangkan Rd adalah *General Purpose Register* (GPR) dari R0 hingga R31. Setelah perintah DEC tersebut dijalankan maka nilai Rd menjadi  $Rd = Rd - 1$ .

Contoh:     LDI R30, 0x8A                     ; R30 = 0x8A  
              DEC R30                         ; R30 menjadi 0x89

### 4. OUT

Perintah OUT atau Out to I/O Location digunakan untuk menyalin nilai dari GPR ke register I/O. Sintaks untuk perintah ini adalah sebagai berikut.

OUT A, Rr

Sedangkan Rr adalah *General Purpose Register* (GPR) dari R0 sampai dengan R31. A adalah register I/O yang beralamat memori ke-0 hingga 63.

Contoh:     OUT PORTD, R11

Perintah OUT tidak dapat digunakan untuk langsung memberikan nilai ke register I/O seperti berikut ini:

Contoh:     OUT DDRC, 0xFE                     ; Salah! Harus lewat Rd dulu  
              Melainkan menyalin dahulu nilai yang akan dikeluarkan ke GPR via perintah LDI, selanjutnya nilai tersebut disalin melalui perintah OUT.

Contoh:     LDI R19, 0xFE                     ; R19 = 0xFE  
              OUT DDRC, R19                     ; DDRC = 0xFE

### 5. RJMP

Perintah RJMP atau Relative Jump digunakan untuk mengarahkan eksekusi program kembali menuju suatu label yang ditunjuk. Keberadaan RJMP dan label pada program membentuk struktur perulangan (*looping*) yang tidak bersyarat dan lazimnya digunakan untuk menjalankan perintah berulang-ulang tanpa henti (*infinite loop*). Sintaks untuk perintah tersebut adalah:



Label:

...

...

RJMP Label

Dengan demikian keberadaan dari RJMP harus selalu disertai dengan label pada program. Keduanya dapat diaplikasikan untuk memberikan keluaran pada pin ATmega16A secara berulang-ulang tanpa henti.

```
Contoh:  LDI R16, 0xFF          ; R16 = 0xFF
          OUT DDRB, R16         ; DDRB = 0xFF
                                   ; Port B menjadi output

Ulang:   LDI R16, 0xF0          ; R16 = 0xF0
          OUT PORTB, R16        ; PB0 s.d. PB3 berisi 0
                                   ; PB4 s.d. PB7 berisi 1

          LDI R16, 0x0F         ; R16 = 0x0F
          OUT PORTB, R16        ; PB0 s.d. PB3 berisi 1
                                   ; PB4 s.d. PB7 berisi 0

          RJMP Ulang
```

#### IV. PROSEDUR PRAKTIKUM

##### A. Percobaan 2-1

1. Buka sebuah project baru, namakan dengan Percobaan 2-1. Ketik listing berikut ini:

```
.nolist
.include "m16def.inc"
.def nilai1 = R18
.def nilai2 = R19
.def tampung = R20
.list
.org 0x00
    LDI nilai1, 0xEA
    LDI nilai2, 0xF1
    MOV tampung, nilai1
    MOV nilai1, nilai2
    MOV nilai2, tampung
```

2. Lakukan kompilasi dan jalankan simulasi. Pada saat simulasi berjalan, isilah nilai dari tiap register untuk per baris instruksi seperti dalam Tabel 7 pada halaman berikutnya.

**Tabel 7. Isi Register untuk Percobaan 2-1**

No.	Instruksi	Isi Register		
		R18	R19	R20
1.	LDI nilai1, 0xEA			
2.	LDI nilai2, 0xF1			
3.	MOV tampung, nilai1			
4.	MOV nilai1, nilai2			
5.	MOV nilai2, tampung			

3. Perhatikan isi register R18 dan R19 di tabel atas tersebut pada instruksi no.2 dan no.5. Apa yang dilakukan oleh program ini?

### B. Percobaan 2-2

1. Buka sebuah project baru, namakan dengan Percobaan 2-2. Ketik listing berikut ini:

```
.nolist
.include "m16def.inc"
.def bilangan1 = R18
.def bilangan2 = R19
.def hasil = R20
.list
.org 0x00
    LDI bilangan1, 0x15
    LDI bilangan2, 0x97
    MOV hasil, bilangan1
    ADD hasil, bilangan2
```

2. Lakukan kompilasi dan jalankan simulasi. Saat simulasi berjalan, isilah nilai dari tiap register untuk per baris instruksi seperti dalam tabel 8.

**Tabel 8. Isi Register untuk Percobaan 2-2**

No.	Instruksi	Isi Register		
		R18	R19	R20
1.	LDI bilangan1, 0x15			
2.	LDI bilangan2, 0x97			
3.	MOV hasil, bilangan1			
4.	ADD hasil, bilangan2			

3. Apa yang dilakukan oleh program tersebut?

### C. Percobaan 2-3

1. Buka sebuah project baru, namakan dengan Percobaan 2-3. Ketik listing berikut ini:

```
.nolist
.include "m16def.inc"
.def nilaiA = R16
.def nilaiB = R17
.list
.org 0x00
    LDI nilaiA, 0x23
    LDI nilaiB, 0xB4
    INC nilaiA
    DEC nilaiB
    DEC nilaiB
```

2. Lakukan kompilasi dan jalankan simulasi. Pada saat simulasi berjalan, isilah nilai dari tiap register untuk per baris instruksi seperti pada tabel 9.

**Tabel 9. Isi Register untuk Percobaan 2-3**

No.	Instruksi	Isi Register	
		R16	R17
1.	LDI nilaiA, 0x23		
2.	<u>LDI nilaiB, 0xB4</u>		
3.	INC nilaiA		
4.	DEC nilaiB		
5.	DEC nilaiB		

3. Apa fungsi dari perintah INC untuk no. 3 dan DEC untuk no.4 - 5 seperti ditandai dalam tabel di atas?

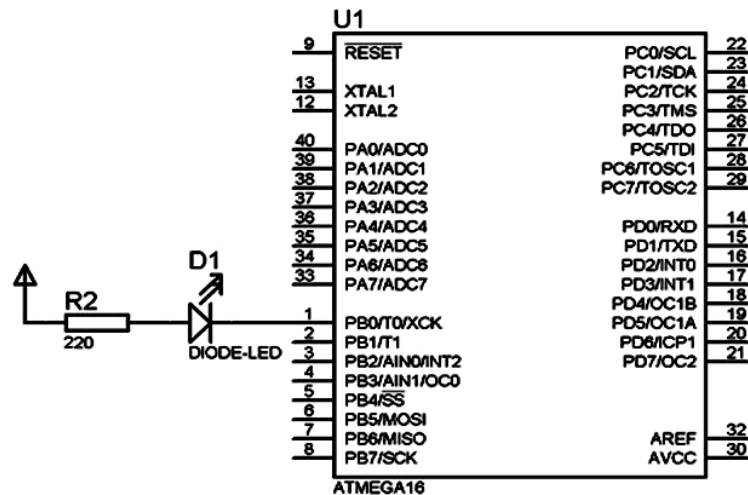
### D. Percobaan 2-4

1. Buka sebuah project baru, namakan dengan Percobaan 2-4.
2. Buatlah sebuah program dengan ketentuan sebagai berikut:
  - a. Menggunakan register R19 dan R20
  - b. Menamai dua register tersebut, masing-masing register1 dan register2
  - c. Register1 diberikan nilai 0x20 dan register2 diberi nilai 0x1C

- d. Turunkan nilai register1 hingga tercapai nilai 0x1D.
  - e. Naikkan nilai register2 hingga tercapai nilai 0x1E
3. Buatlah tabel instruksi dan isi register untuk program yang anda buat tersebut!

### E. Percobaan 2-5

1. Buatlah rangkaian seperti berikut ini:



Gambar 16. Rangkaian untuk Percobaan 2-5

2. Buka sebuah project baru, namakan dengan Percobaan 2-5. Ketik listing berikut ini:

```
;Menyalakan LED
.nolist
.include "m16def.inc"
.list
.org 0x00
    LDI R21, 0xFF
    OUT DDRB, R21
MULAI:
    LDI R21, 0xFE
    OUT PORTB, R21
    RJMP MULAI
```

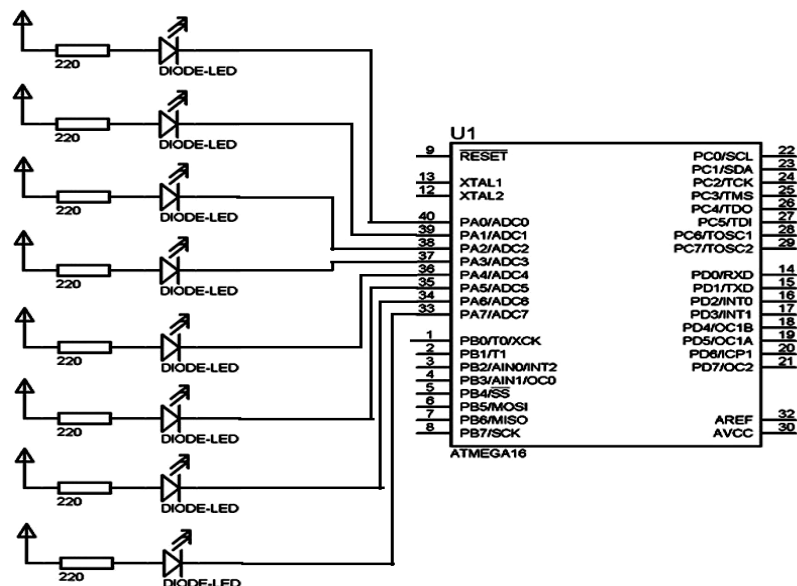
3. Lakukan kompilasi dan jalankan simulasi. Pada saat simulasi berjalan, isilah nilai dari tiap register untuk per baris instruksi seperti dalam tabel di halaman berikutnya.

**Tabel 10. Isi Register untuk Percobaan 2-5**

No.	Instruksi	Isi Register		
		R21	DDRB	PORTB
1.	LDI R21, 0xFF			
2.	OUT DDRB, R21			
3.	LDI R21, 0xFE			
4.	OUT PORTB, R21			
5.	RJMP MULAI			

**F. Percobaan 2-6**

1. Buatlah rangkaian seperti berikut ini:

**Gambar 17. Rangkaian untuk Percobaan 2-6**

2. Buka sebuah project baru, namakan dengan Percobaan 2-6. Ketik listing berikut ini:

```
;Mengatur penyalan 8 buah LED
.nolist
.include "m16def.inc"
.list
.org 0x00
    LDI R21, 0xFF
    OUT DDRA, R21
MULAI:
    LDI R21, 0x55
    OUT PORTA, R21
    RJMP MULAI
```

3. Lakukan kompilasi dan jalankan simulasi. Pada saat simulasi berjalan, isilah nilai dari tiap register untuk per baris instruksi seperti dalam tabel di halaman berikut ini.

**Tabel 11. Isi Register untuk Percobaan 2-5**

No.	Instruksi	Isi Register		
		R21	DDRA	PORTA
1.	LDI R21, 0xFF			
2.	OUT DDRA, R21			
3.	LDI R21, 0xAA			
4.	OUT PORTA, R21			
5.	RJMP MULAI			

## V. TUGAS PEMROGRAMAN

1. Buatlah sebuah program dengan ketentuan sebagai berikut:
  - a. Gunakan dua buah register dari R0 s.d. R15. Namakan register1 dan register2.
  - b. Gunakan satu buah register dari R16 s.d. R31. Namakan dengan tampung.
  - c. Masukkan nilai 0x24 ke register1 dan 0x38 ke register2 dengan perantara register tampung.
  - d. Naikkan nilai register1 sebanyak tiga kali dan turunkan nilai register2 dua kali.
  - e. Tukarkan nilai register1 dan register2 dengan perantara register tampung.
  - f. Jumlahkan tampung dengan register1.
  - g. Jumlahkan tampung dengan register2.
2. Buatlah sebuah program yang menyalakan LED di port A dan port C, masing-masing dengan pola seperti pada Gambar 18 di halaman berikutnya.

**Port A**

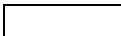
PA7	PA6	PA5	PA4	PA3	PA2	PA1	PA0

**Port C**

PC7	PC6	PC5	PC4	PC3	PC2	PC1	PC0

**Keterangan:**

 = Mati

 = Menyala

**Gambar 18. Gambar untuk Soal No. 2.**

**CATATAN:**

Pada kedua program tersebut, buatlah tabel instruksi dan isi register sesuai hasil simulasi. Sertakan pula gambar diagram alirnya!

## UNIT III

### Kondisi Percabangan dan Input Digital

#### I. TUJUAN PRAKTIKUM

1. Mahasiswa mampu membuat program untuk kondisi percabangan, baik tunggal maupun majemuk.
2. Mahasiswa mampu menerapkan kondisi percabangan dalam pembuatan delay
3. Mahasiswa mampu merakit rangkaian untuk menjalankan operasi input digital
4. Mahasiswa mampu membuat program untuk melakukan operasi input digital

#### II. ALAT DAN BAHAN

1. 1 unit laptop (dari peserta praktikum) atau 1 unit PC (dari Lab. Komputer).
2. Perangkat lunak ATMEL Studio 6 dan ExtremeBurner AVR 1.4.2
3. 1 set minimum system ATmega16
4. 1 set kabel male-to-female
5. 1 buah LED 5mm
6. 1 buah resistor 220  $\Omega$
7. 1 buah saklar tekan
8. 1 buah buzzer
9. Lembar Pencatatan

#### III. DASAR TEORI

##### A. Komponen Elektronika

Beberapa komponen tambahan yang digunakan untuk praktikum modul ketiga adalah meliputi:

1. *Buzzer*

*Buzzer* merupakan perangkat elektronik yang digunakan untuk memberikan keluaran dalam bentuk suara. Perangkat ini aktif dan akan bersuara jika diberikan tegangan masukan dengan level tertentu. Terdapat beragam jenis *buzzer*, namun yang sering digunakan dalam eksperimen atau proyek elektronika sederhana adalah *buzzer* berukuran kecil dengan penutup plastik yang menggunakan tegangan masukan +5V. Kutub positif dari *buzzer* tersebut ditandai oleh simbol  $\oplus$  dan biasanya memiliki kaki yang lebih panjang daripada kaki pada kutub negatif.



Pada umumnya buzzer langsung dihubungkan dengan pin output pada mikropengendali yang akan membunyikannya. Buzzer dalam rangkaian semacam itu akan berbunyi dengan volume yang maksimal. Untuk mengurangi volume selain dengan menggunakan instruksi pemrograman, buzzer juga dapat dirangkai seri dengan resistor sehingga membatasi arus yang mengalir kepadanya.



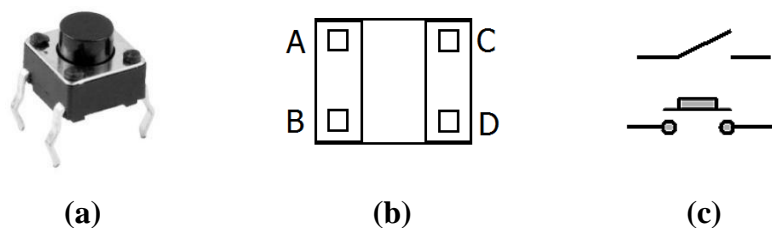
**Gambar 19. Buzzer (a) Bentuk Fisik (b) Simbol Komponen**

## 2. Pushbutton

*Pushbutton* atau saklar tekan adalah komponen elektronika yang digunakan untuk memutuskan atau menghubungkan dua jalur rangkaian. *Pushbutton* yang disebut juga sebagai *tactile switch* ini termasuk jenis saklar *Single Pole Single Throw* (SPST) dengan simbol komponen seperti pada Gambar 20 (c) atas atau bawah. Pada kondisi tidak ditekan, maka kedua jalur yang ada menjadi terbuka (*normally open*), tidak terhubung.

Penampang bawah dari *pushbutton* tersebut adalah seperti dalam Gambar 19 (b). Kaki A dan B saling terhubung, demikian pula kaki C dan D. Saat tidak ditekan, lajur A – B dan C – D tidak terhubung. Kedua lajur tersebut menjadi terhubung saat tombol ditekan.

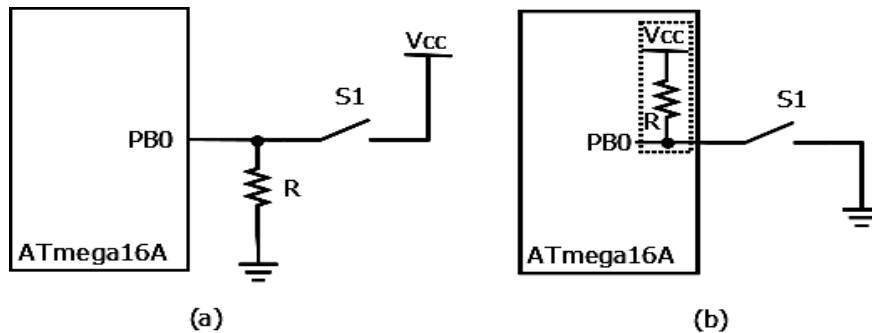
Pada umumnya untuk aplikasi mikropengendali, *pushbutton* digunakan sebagai perangkat masukan. *Pushbutton* tersebut dihubungkan dengan  $V_{CC}$  atau *ground* di satu sisi dan ke pin mikropengendali di sisi lain. Sehingga ketika ditekan terdapat tegangan masukan 5 V atau 0 V yang kemudian oleh mikropengendali dianggap sebagai logika 1 atau 0.



**Gambar 20. Pushbutton (a) Bentuk Fisik (b) Penampang Bawah (c) Simbol Komponen**

## B. Rangkaian Input

Sama halnya dengan rangkaian output, terdapat dua jenis rangkaian input yaitu *Active High* dan *Active Low*. Pada contoh berikut dalam Gambar 21, pin PB0 digunakan untuk menerima masukan dari *pushbutton* S1.



Gambar 21. Rangkaian Input (a) *Active High* (b) *Active Low*

Pada rangkaian input jenis *Active High*, terdapat resistor R yang berfungsi sebagai *pull-down*. Umumnya R tersebut bernilai 10 K $\Omega$ . Pemberian resistor *pull-down* memiliki tujuan untuk menghilangkan kondisi mengambang (*floating*) saat S1 tidak ditekan. Gejala tersebut ditandai dengan nilai masukan pada PB0 yang tidak pasti, berubah-ubah antara 0 dan 1. Bila S1 tidak ditekan, kontak terbuka, maka resistor tersebut akan menghubungkan PB0 dengan *ground* sehingga terdeteksi nilai logika 0. Saat S1 ditekan (kondisi aktif), kontak tertutup, maka PB0 akan terhubung dengan V<sub>CC</sub>, sehingga terdeteksi nilai logika 1.

Pada rangkaian input jenis *Active Low*, pushbutton S1 dihubungkan dengan *ground* seperti pada Gambar 21 (b). Untuk menghindari gejala *floating*, saat S1 tidak ditekan maka perlu diberikan resistor *pull-up*. Meski demikian, resistor tersebut tidak perlu diberikan secara eksternal, melainkan cukup diaktifkan saja dari pin PB0, mengingat setiap pin I/O untuk ATmega16A sudah disertai resistor *pull-up* internal. Dengan adanya resistor tersebut maka saat S1 tidak ditekan, PB0 akan terhubung dengan V<sub>CC</sub> sehingga terdeteksi nilai logika 1. Kemudian bila S1 ditekan (kondisi aktif), maka PB0 akan terhubung dengan *ground* sehingga terdeteksi nilai logika 0.

## C. Register-Register Pengatur Fungsi Input

Pengaturan register-register I/O perlu dilakukan agar pin-pin I/O yang terdapat pada mikropengendali dapat difungsikan sebagai input. Register-register yang dimaksud tersebut adalah sebagai berikut:

### 1. DDRx

Sebagaimana pada rangkaian output, fungsi suatu pin sebagai input juga diatur melalui register DDRx, yaitu dengan memberikan nilai 0 kepada pin tersebut. Penjelasan lebih lanjut tentang register ini sudah terdapat pada unit 2.

Contoh:     LDI R27, 0                                 ; R27 = 0  
               OUT DDRC, R27                         ; DDRC = 0. Semua pin di port C  
    ; menjadi input

### 2. PORTx

Pada rangkaian output, register PORTx bertugas untuk memberikan nilai logika 0 atau 1 kepada suatu pin. Meskipun demikian jika register PORTx untuk suatu pin yang berfungsi sebagai masukan diberikan nilai 1, maka ini akan mengaktifkan resistor *pull-up* internal pada pin tersebut. Sebagaimana penjelasan pada poin B di Dasar Teori, resistor ini digunakan untuk meniadakan gejala *floating*.

Contoh:     LDI R27, 0x00                             ; R27 = 0x00  
               OUT DDRC, R27                         ; DDRC = 0x00. Semua pin di port C  
    ; menjadi input  
  
               LDI R27, 0xFF                           ; R27 = 0xFF  
               OUT PORTC, R27                        ; Internal pull-up di semua pin  
    ; diaktifkan

Register DDRx dan PORTx memiliki peranan untuk mengatur fungsi-fungsi input dan output pada suatu pin. Selengkapnya tentang kombinasi nilai dari DDRx dan PORTx untuk suatu pin I/O adalah seperti tertera dalam tabel 12 berikut.

**Tabel 12. Nilai DDRx dan PORTx untuk Suatu Pin I/O**

Fungsi Pin	Nilai Register		Internal Pull-Up	Keluaran
	DDRx	PORTx		
Output	1	0	Tidak Berfungsi	0
	1	1	Tidak Berfungsi	1
Input	0	0	Non Aktif	-
	0	1	Aktif	-

### 3. PINx

PINx adalah register 8 bit yang berfungsi untuk menyimpan nilai masukan dari suatu pin. Oleh karenanya PINx hanya digunakan pada operasi input. Nilai yang diperoleh

dari pin-pin dalam suatu port tersebut selanjutnya dapat disalin ke GPR melalui perintah IN. Serupa halnya dengan dua register sebelumnya, terdapat empat register PINx untuk mikropengendali ATmega16A, yaitu PINA, PINB, PINC dan PIND.

Pendeteksian secara terus menerus isi register PINx pada umumnya menggunakan perintah perulangan. Sebelum ini dilaksanakan, pada program perlu adanya pendefinisian fungsi I/O melalui register DDRx dan PORTx terlebih dahulu.

```
Contoh:  LDI R18, 0x00          ; R18 = 0x00
          OUT DDRA, R18        ; Semua pin di port A sebagai input
          LDI R18, 0xFF        ; R18 = 0xFF
          OUT PORTA, R18       ; Mengaktifkan internal pull-up
Kembali: IN R18, PINA          ; Menyalin nilai dari Port A ke R18
          MOV R0, R18          ; Menyalin nilai dari R18 ke R0
          RJMP Kembali
```

#### D. Instruksi-Instruksi untuk Perulangan, Percabangan dan Input

##### 1. NOP

NOP atau *No Operation* adalah instruksi yang memerintahkan CPU untuk berhenti bekerja selama satu siklus mesin. Sintaks untuk perintah ini:

##### NOP

Pada umumnya perintah ini digunakan dalam sebuah struktur perulangan dengan tujuan sekadar untuk membuang satu siklus mesin. Dalam program berikut ini pin PB0 berubah-ubah dari 1 ke 0 dan sebaliknya dimana perintah NOP menghentikan operasi selama satu siklus mesin saat terjadi pergantian kondisi di PB0 tersebut.

```
Contoh:  LDI R17, 0xFF          ; R17 = 0xFF
          OUT DDRB, R17         ; Port B menjadi output
Ulang:   OUT PORTB, R17         ; PORTB = 0xFF
          DEC R17               ; R17 = 0xFE
          NOP                   ; Tidak ada operasi
          OUT PORTB, R17        ; PORTB = 0xFE
          INC R17               ; R17 = 0xFF
          RJMP Ulang            ; Kembali ke label Ulang
```

Sesaat setelah perintah NOP dieksekusi, kondisi program dalam hal ini nilai-nilai register, tidak mengalami perubahan apa pun. Oleh karena itu perintah NOP yang dikombinasikan dengan struktur perulangan untuk nilai tertentu dapat dimanfaatkan dalam pembuatan waktu jeda (*delay*) untuk program.

## 2. BRNE

BRNE atau *Branch If Not Equal* adalah instruksi percabangan yang akan mengecek nilai dari sebuah GPR di instruksi sebelumnya apakah sama dengan nol. Jika ya, maka instruksi berikutnya akan diakses, namun jika tidak maka alur program akan berpindah ke label yang ditunjuk. Instruksi ini umumnya digunakan dalam sebuah perulangan yang mempersyaratkan nilai dari sebuah GPR sebesar nol terlebih dahulu sebelum instruksi selanjutnya dijalankan. Sintaks untuk perintah tersebut adalah sebagaimana berikut:

**Label1:**

...

**BRNE Label1**

Sebenarnya ketika sebuah GPR menjalani perintah INC atau DEC, terdapat register penanda (*flag register*) bernama Z yang akan mengecek apakah nilai GPR tersebut sama dengan nol. Jika ya maka  $Z = 0$ , tapi bila tidak maka  $Z = 1$ . Perintah BRNE selanjutnya mengecek nilai dari register Z tersebut untuk menentukan alur program.

Program berikut ini menghitung mundur register R21 dari 7 hingga 0 dengan menampilkannya ke Port C.

```
Contoh:  LDI R21, 0xFF          ; R21 = 0xFF
          OUT DDRC, R21        ; Port C menjadi output
Awal:    LDI R21, 0x07          ; R21 = 0x07
Ulang:   OUT PORTC, R21        ; Keluarkan nilai R21 ke port C
          DEC R21               ; R21 = R21 - 1
          BRNE Ulang           ; Jika R21 != 0 ke Ulang
          OUT PORTC, R21        ; PORTC = 0x00
          RJMP Awal            ; Jika R21 = 0 ke Awal
```

## 3. CPI

CPI atau *Compare with Immediate* adalah instruksi yang digunakan untuk membandingkan isi suatu GPR dengan suatu nilai. Hasil dari perbandingan tersebut adalah berupa suatu keputusan apakah nilainya sama, lebih besar, kurang dari, sama dengan nol atau tidak sama dengan nol. CPI lazimnya digunakan secara berpasangan dengan instruksi percabangan bersyarat (*branch*). Pada praktikum ini, instruksi percabangan tersebut adalah BRNE. Sintaks penulisan dari CPI adalah seperti berikut:

**CPI Rd, K**

Dengan Rd adalah GPR dari R16 hingga R31 dan K menyatakan nilai yang dapat berupa bilangan biner, heksadesimal, oktal maupun desimal. Bila digunakan

berpasangan dengan BRNE, maka CPI akan mengecek apakah nilai  $Rd = K$ . Jika tidak, maka alur program akan beralih ke label yang ditunjuk oleh instruksi BRNE, namun jika sama, maka instruksi sesudah BRNE yang akan dijalankan.

Pada program berikut ini diperagakan penambahan nilai register R16 satu demi satu (*increment*) dari 0x18 hingga 0x1D secara berulang-ulang.

Contoh:

```
Mulai:    LDI R16, 0x18           ; R16 = 0x18
Ulang:    INC R16                ; R16 = R16 + 1
          CPI R16, 0x1D          ; Apakah R16 = 0x1D?
          BRNE Ulang            ; Jika tidak, ke Ulang
          RJMP Mulai            ; Jika ya, ke Mulai
```

#### 4. IN

IN atau *In from I/O Location* adalah perintah yang berfungsi untuk menyalin isi dari nama atau alamat register I/O ke GPR. Perintah ini lazimnya digunakan untuk operasi input, saat hendak mengambil nilai masukan dari suatu port. Sintaks penulisannya:

**IN Rd, A**

Dengan Rd adalah nama GPR tujuan, dari R0 hingga R31 sedangkan A adalah nama atau alamat register I/O. Selengkapnya tentang nama dan alamat dari beberapa register I/O adalah seperti dalam Tabel 13 berikut.

**Tabel 13. Nama dan Alamat Register I/O untuk Port A, B, C dan D**

Port	Register I/O	
	Nama	Alamat
A	PINA	0x19
	DDRA	0x1A
	PORTA	0x1B
B	PINB	0x16
	DDRB	0x17
	PORTB	0x18
C	PINC	0x13
	DDRC	0x14
	PORTC	0x15
D	PIND	0x10
	DDRD	0x11
	PORTD	0x12

Contoh: `IN R10, PORTB` ; Menyalin input dari Port B ke R10  
 Alternatifnya nama register PORTB tersebut dapat diganti dengan alamatnya yaitu seperti berikut ini:

Contoh: `IN R10, 0x18` ; Menyalin input dari alamat Port B  
 ; ke dalam R10

Jika instruksi IN digabungkan dengan perintah perulangan tak bersyarat, maka isi dari port input dapat dicek secara terus-menerus. Pada program di bawah ini diperlihatkan pengecekan terhadap nilai dari port A dan B sebagai input. Nilai tersebut selanjutnya dijumlahkan dan ditampilkan di R2.

Contoh: `LDI R16, 0x00` ; R16 = 0x00  
`OUT DDRA, R16` ; Port A menjadi input  
`OUT DDRB, R16` ; Port B menjadi input  
`OUT PORTA, R16` ; Pull-up port A dinonaktifkan  
`OUT PORTB, R16` ; Pull-up port B dinonaktifkan

Ulang: `IN R1, PORTA` ; Menyalin input port A ke R1  
`IN R2, PORTB` ; Menyalin input port B ke R2  
`ADD R2, R1` ; R2 = R2 + R1  
`RJMP Ulang` ; Kembali ke Ulang

## IV. PROSEDUR PRAKTIKUM

### A. Percobaan 3-1

1. Buka sebuah project baru, namakan dengan Percobaan 3-1. Ketik listing berikut:

```
.nolist
#include "m16def.inc"
.list
.org 0x00
    RJMP mulai
mulai:
    LDI R16, 0x05
    LDI R17, 0x00
ulang:
    DEC R16
    NOP
    BRNE ulang
    INC R17
    RJMP mulai
```

2. Lakukan kompilasi dan jalankan simulasi. Pada saat simulasi berjalan, isilah nilai dari tiap register untuk per baris instruksi seperti dalam tabel berikut:

**Tabel 14. Isi Register untuk Percobaan 3-1**

No.	Instruksi	Isi Register				
		R16				
1.	RJMP mulai					
2.	mulai: LDI R16, 0x05					
3.	LDI R17, 0x00					
4.	ulang: DEC R16 NOP BRNE ulang	Perulangan				
		1	2	3	4	5
5.	INC R17					
6.	RJMP mulai					

3. Apa perbedaan antara instruksi RJMP dengan BRNE?

### B. Percobaan 3-2

1. Buka sebuah project baru, namakan dengan Percobaan 3-2. Ketik listing berikut:

```
.nolist
.include "m16def.inc"
.list
.org 0x00
    RJMP mulai
mulai:
    LDI R16, 0x00
    LDI R17, 0x03
ulang1:
    LDI R18, 0x02
ulang:
    DEC R18
    NOP
    BRNE ulang
    DEC R17
    BRNE ulang1
    INC R16
    RJMP mulai
```

2. Lakukan kompilasi dan jalankan simulasi. Pada saat simulasi berjalan, isilah nilai dari tiap register untuk per baris instruksi seperti dalam tabel 15 pada halaman berikutnya.



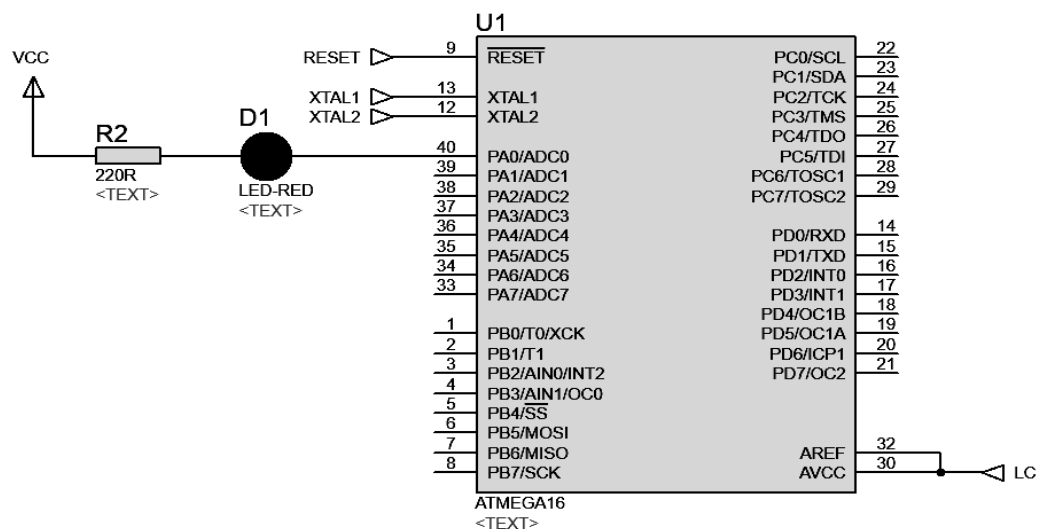
**Tabel 15. Isi Register untuk Percobaan 3-2**

No.	Instruksi	Isi Register		
		R16	R17	R18
1.	RJMP mulai			
2.	mulai: LDI R16, 0x00			
3.	LDI R17, 0x03			
4.	ulang1: LDI R18, 0x02			
5.	ulang: DEC R18 NOP BRNE ulang			Perulangan
				1      2
6.	DEC R17 BRNE ulang1		Perulangan	
			1      2      3	
7.	INC R16			
8.	RJMP mulai			

3. Apa yang dilakukan oleh program tersebut?

### C. Percobaan 3-3

1. Buatlah rangkaian seperti gambar berikut ini:

**Gambar 22. Rangkaian untuk Percobaan 3-3**

2. Buka sebuah project baru, namakan dengan Percobaan 3-3. Ketik kode program seperti berikut ini:

```
.nolist
.include "m16def.inc"
.list
.def register_output = R20
.equ nilai_perulangan_r21 = 250
.equ nilai_perulangan_r22 = 200
.org 0x00
    RJMP mulai

mulai:
    ;Port A sebagai output
    LDI register_output, 0xFF
    OUT DDRA, register_output

ulang:
    ;LED di PA0 dinyalakan
    LDI register_output, 0xFE
    OUT PORTA, register_output

    ;Fungsi delay
    LDI R22, nilai_perulangan_r22
ulang4:
    LDI R21, nilai_perulangan_r21
ulang3:
    NOP
    DEC R21
    BRNE ulang3
    DEC R22
    BRNE ulang4

    ;LED di PA0 dimatikan
    LDI register_output, 0xFF
    OUT PORTA, register_output

    ;Fungsi delay
    LDI R22, nilai_perulangan_r22
ulang2:
    LDI R21, nilai_perulangan_r21
ulang1:
    NOP
    DEC R21
    BRNE ulang1
    DEC R22
    BRNE ulang2

    ;Kembali ke awal
    RJMP ulang
```

4. Simpan, lakukan kompilasi dan unduh program ke mikropengendali melalui ExterneBurner AVR. Amati hasilnya pada rangkaian.

5. Dalam tabel berikut ini pada kondisi 1, isilah nilai untuk R21 dan R22. Semua dengan nilai yang lebih kecil dari kondisi awal. Kalikan nilai keduanya untuk memperoleh total perulangan. Ganti `nilai_perulangan_r21` dan `nilai_perulangan_r22` pada baris program berikut ini dengan nilai seperti apa yang dituliskan dalam tabel.

```
.equ nilai_perulangan_r21 = 250
.equ nilai_perulangan_r22 = 200
```

**Tabel 16. Isi Register untuk Percobaan 3-3**

No.	Instruksi	Nilai Perulangan		
		R21 (a)	R22 (b)	Total (a x b)
1.	Kondisi Awal	250	200	50000
2.	Kondisi 1			
3.	Kondisi 2			
4.	Kondisi 3			
5.	Kondisi Akhir	50	25	1250

6. Jalankan kompilasi, unduh ke mikropengendali dan amati kondisi berkedip atau *blinking* pada LED.
7. Ulangi langkah ke-4 dan 5 di atas untuk kondisi 2 dan 3 (dengan nilai-nilai yang lebih kecil dari kondisi sebelumnya)
8. Apa yang dapat anda simpulkan dengan hubungan antara total perulangan terhadap kondisi berkedip atau *blinking* pada LED?

#### D. Percobaan 3-4

1. Buatlah rangkaian seperti pada gambar 23 di halaman berikutnya
2. Buka sebuah project baru, namakan dengan Percobaan 3-4. Ketik listing berikut:

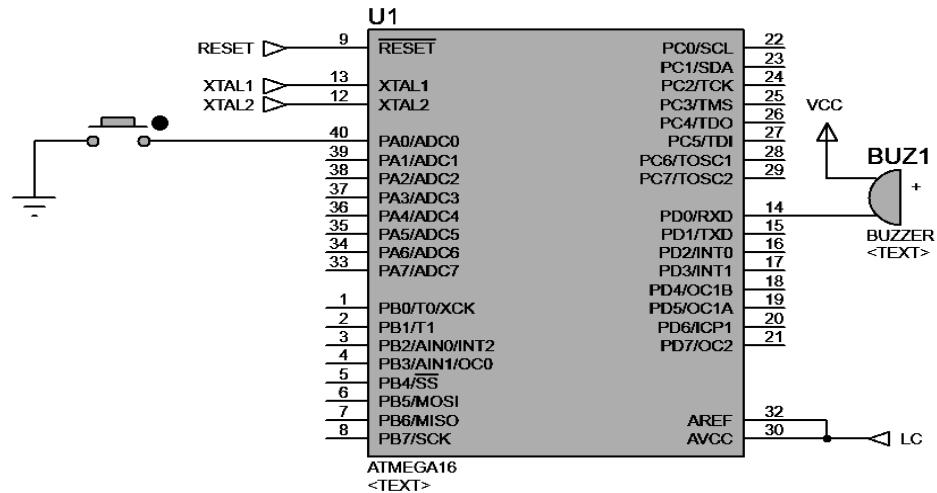
```
.nolist
.include "m16def.inc"
.list
.org 0x00
    LDI R16, 0x00
    OUT DDRA, R16
    LDI R16, 0xFF

    ;perintah di bawah ini untuk
    ;mengaktifkan internal pull-up
    OUT PORTA, R16
```

```

    OUT DDRD, R16
ulang:
    IN R16, PINA
    OUT PORTD, R16
    RJMP ulang

```



Gambar 23. Rangkaian untuk Percobaan 3-4

4. Lakukan kompilasi dan jalankan simulasi. Isilah Tabel 17 berikut ini.

Tabel 17. Isi Register untuk Percobaan 3-4

No.	Instruksi	Isi Register					
		R16	DDRA	PORTA	PINA	DDRD	PORTD
1.	LDI R16, 0x00						
2.	OUT DDRA, R16						
3.	LDI R16, 0xFF						
4.	OUT PORTA, R16						
5.	OUT DDRD, R16						
(Jika tombol ditekan)							
6.	ulang: IN R16, PINA						
7.	OUT PORTD, R16						
8.	RJMP ulang						
(Jika tombol tidak ditekan)							
6.	ulang: IN R16, PINA						
7.	OUT PORTD, R16						
8.	RJMP ulang						

5. Pindahkan kabel di PD0 ke PD1. Tekan saklar di PA0. Apa yang terjadi? Sekarang pindahkan juga kabel dari PA0 ke PA1. Tekan saklar di PA1. Apakah buzzer berbunyi? Mengapa demikian?

### E. Percobaan 3-5

1. Masih menggunakan rangkaian yang sama seperti pada percobaan 3-4, sekarang buka project baru, namakan dengan Percobaan 3-5. Ketik kode berikut ini:

```
.nolist
.include "m16def.inc"
.list
.org 0x00
    LDI R16, 0x00
    OUT DDRA, R16
    LDI R16, 0xFF

    ;perintah di bawah ini untuk
    ;mengaktifkan internal pull-up
    OUT PORTA, R16

    OUT DDRD, R16
    OUT PORTD, R16

ulang:
    IN R16, PINA

    ;Bandingkan nilai di R16
    ;dengan 0xFE
    CPI R16, 0xFE

    ;Jika tidak sama maka ke
    ;label non_aktif
    BRNE non_aktif

    ;Jika sama maka masukkan
    ;nilai 0x00 ke R16
    LDI R16, 0x00
    RJMP keluaran

non_aktif:
    ;Jika tidak sama maka
    ;masukkan nilai 0xFF ke R16
    LDI R16, 0xFF

keluaran:
    ;R16 = 0x00 -> Buzzer berbunyi
    ;R16 = 0xFF -> Buzzer diam
    OUT PORTD, R16

    RJMP ulang
```

2. Lakukan kompilasi dan jalankan simulasi. Isilah Tabel 18 berikut ini.

**Tabel 18. Isi Register untuk Percobaan 3-4**

No.	Instruksi	Isi Register					
		R16	DDRA	PORTA	PINA	DDRD	PORTD
1.	LDI R16, 0x00						
2.	OUT DDRA, R16						
3.	LDI R16, 0xFF						
4.	OUT PORTA, R16						
5.	OUT DDRD, R16						
6.	OUT PORTD, R16						
<b>(Jika tombol tidak ditekan)</b>							
7.	ulang: IN R16, PINA						
8.	CPI R16, 0xFE						
9.	BRNE non_aktif						
10.	non_aktif: LDI R16, 0xFF						
11.	keluaran: OUT PORTD, R16						
12.	RJMP ulang						
<b>(Jika tombol ditekan)</b>							
7.	ulang: IN R16, PINA						
8.	CPI R16, 0xFE						
9.	BRNE non_aktif						
10.	LDI R16, 0x00						
11.	RJMP keluaran						
12.	keluaran: OUT PORTD, R16						
13.	RJMP ulang						

3. Pindahkan kabel di PD0 ke PD1. Tekan saklar di PA0. Apa yang terjadi? Pindahkan lagi kabel ke PD2 dan seterusnya dengan menekan saklar PA0 sesudahnya. Amati hasilnya. Apa perbedaan Percobaan3-5 ini dengan sebelumnya?

## V. TUGAS PEMROGRAMAN

1. Buatlah sebuah program dengan ketentuan sebagai berikut:
  - a. Pilih satu buah register dari R16 s.d. R31. Namakan dengan hitung.
  - b. Isikan nilai hitung dengan nilai 0x0A.
  - c. Buatlah label bernama turun dan turunkan nilai hitung tersebut sebanyak 1.
  - d. Berikan perintah tidak aktif (*No Operation*).
  - e. Cek apakah nilai hitung sudah mencapai nol. Jika belum maka kembali ke label turun (seperti dalam butir c di atas)
  - f. Jika sudah, maka masukkan nilai 0x05 ke hitung.
  - g. Buatlah label bernama naik dan naikan nilai hitung tersebut sebanyak 1.
  - h. Berikan perintah tidak aktif (*No Operation*).
  - i. Cek apakah nilai hitung sudah mencapai 0x0A. Jika belum maka kembali ke label naik (seperti dalam butir g di atas).
  - j. Jika sudah, maka program kembali ke label turun (seperti dalam butir c di atas).
2. Dengan mengembangkan kode dari Percobaan3-5, buatlah program yang melakukan hal berikut ini:
  - a. Jika tombol ditekan, maka LED mati dan buzzer berbunyi.
  - b. Jika tombol dilepas, maka LED hidup dan buzzer diam.

Program menggunakan logika *active low*, yaitu perangkat akan aktif jika diberikan logika nol. Port dimana tombol, LED dan buzzer terletak bersifat bebas. Gunakan juga satu buah register yang dapat dipilih secara bebas dari R16 s.d. R31.

### Petunjuk:

1. Pada soal nomor 1, gunakan perintah CPI dan BRNE untuk instruksi pada butir i. Cek kembali pada kode program Percobaan3-5 yang menerapkan penggunaan kedua perintah tersebut.
2. Pada soal nomor 2, untuk menguji apakah kode yang ditulis sudah benar atau belum, gunakan fasilitas simulasi di dalam ATME Studio. Tidak perlu menggunakan perangkat yang sesungguhnya.
3. Baik untuk soal nomor 1 dan 2, tulis kode program dan gambarkan diagram alirnya!