Dasar Assembler AVR

Salah satu alasan Bahasa assembler digunakan karena kita ingin melihat efeknya terhadap isi register dan memory. Bagi pelajar atau mahasiswa yang sedang mempelajari arsitektur komputer atau microprosessor sebaiknya menggunakan bahasa Asembler selain mempelajari bahasa C .

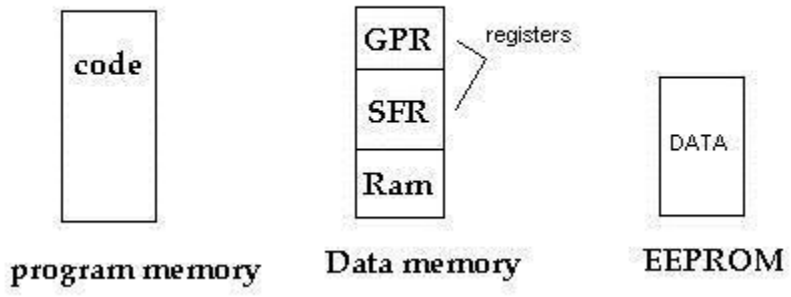
Untuk membuat program assembler kita harus mengetahui peta memori, register-register dan arsitektur AVR. Register digunakan untuk menyimpan data sementara fungsinya mirip spt RAM tapi register biasanya diakses dgn nama buka lokasi alamat spt mengakses RAM.

Peta Memori

Memori di AVR terbagi menjadi 2 bagian:

1. Program memori

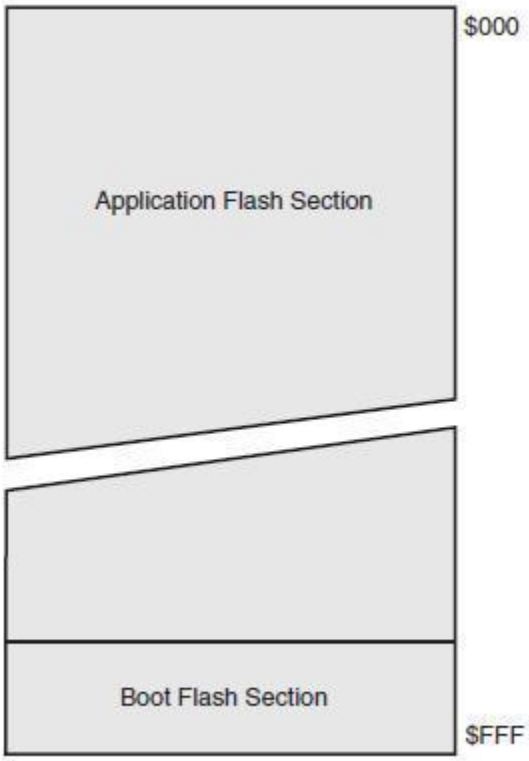
2. Data Memori (Register umum(GPR),Register khusus(SFR), RAM dan EEPROM)



Memori AVR

1. Program Memory

ATMega8535 memiliki On-Chip In-System Reprogrammable Flash Memory untuk menyimpan program. Untuk alasan keamanan, program memory dibagi menjadi dua bagian yaitu Boot Flash Section dan Application Flash Section. Boot Flash Section digunakan untuk menyimpan program Boot Loader, yaitu program yang harus dijalankan pada saat AVR reset atau pertama kali diaktifkan. Application Flash Section digunakan untuk menyimpan program aplikasi yang dibuat user. AVR tidak dapat menjalankan program aplikasi ini sebelum menjalankan program Boot Loader.

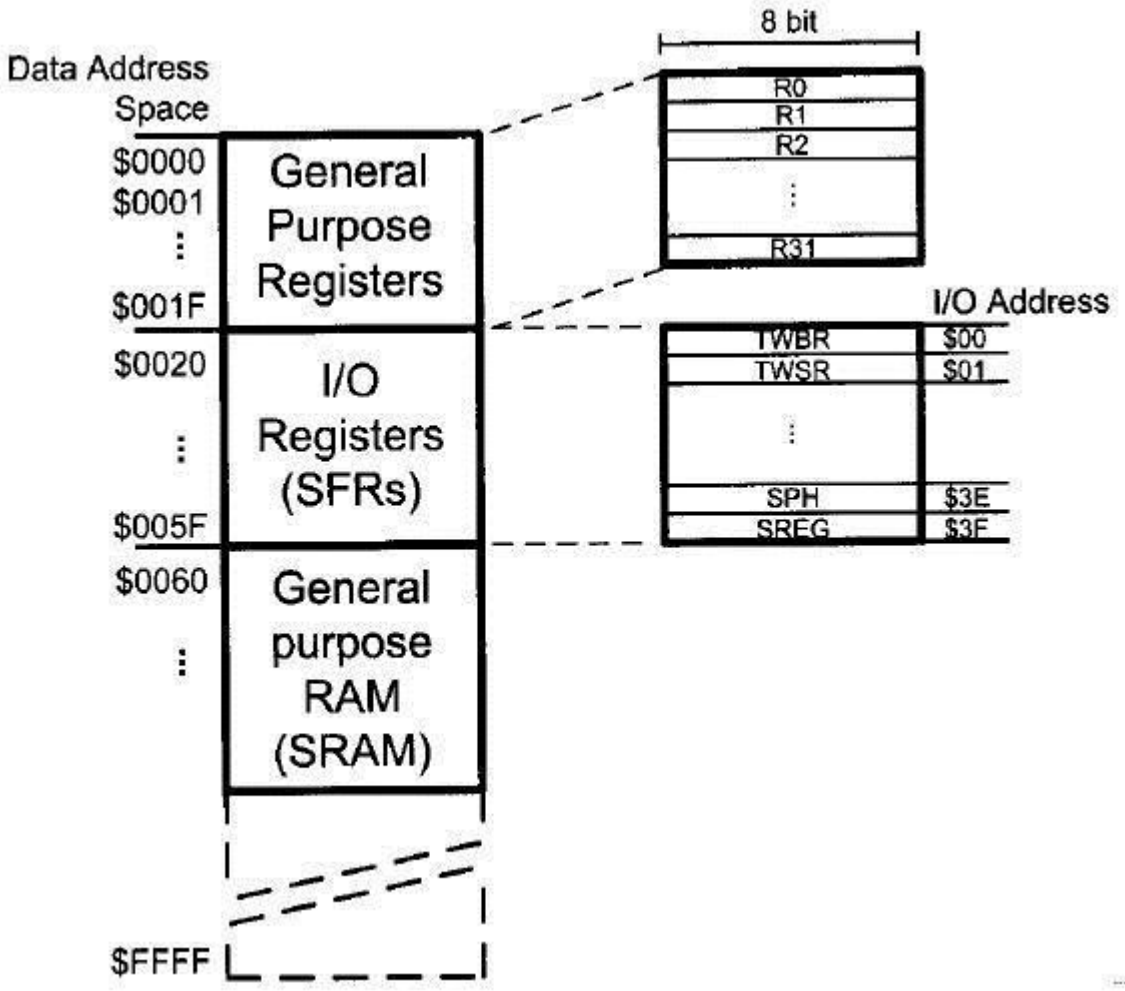


Program Memory Map

2. Data Memory

Data Memory AVR di alokasikan untuk :

1. Register File (GPR, general purpose reg), terdiri dari 32 register.
2. I/O register (SFR,special purpose reg) terdiri dari 64 register.
3. Internal data SRAM.



memory-register

2.a General Purpose Register (GPR)

AVR mempunya 32 register untuk menyimpan data sementara register tersebut adalah R0 sampai R31 berada di lokasi memory terendah (00H~1FH). Register ini berlaku seperti register Accumulator pada microprosessor lain. Register ini digunakan untuk operasi logika dan aritmetik

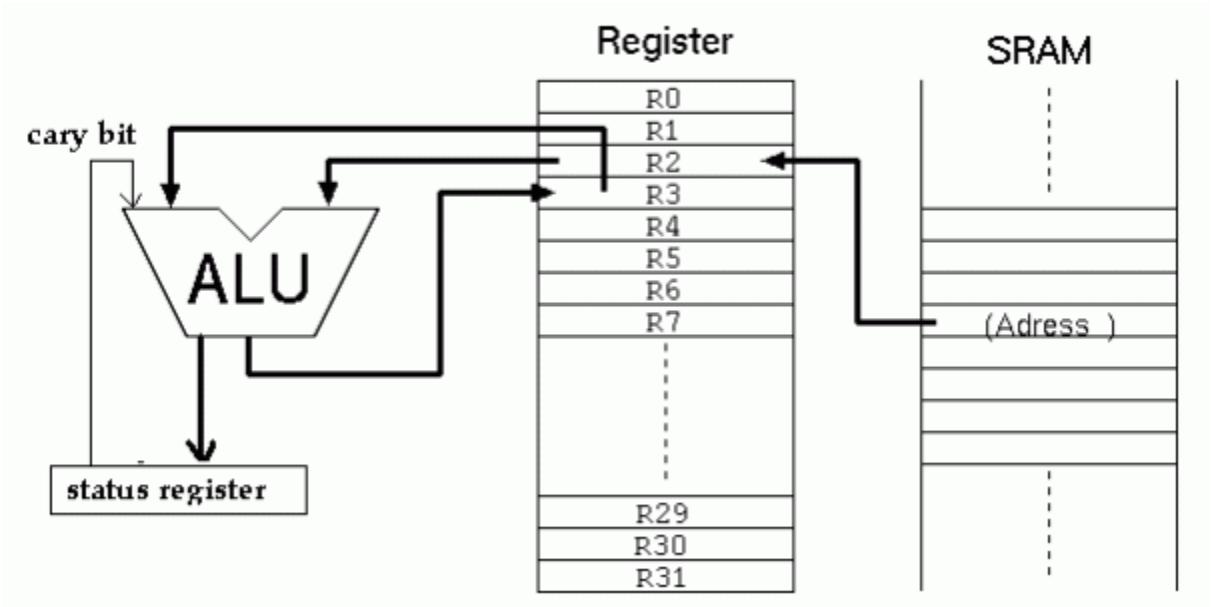
***2.b IO register* (SFR)**

Dialokasikan untuk register2 fungsi khusus spt register untuk Timer, ADC ,IO port,UART, dll. contoh register dilokasi ini: DDRA,DDRB,PORTA,PINA,UCSRA dll

2.c General Purpose RAM

RAM adalah tempat menyimpan data umum yang tidak bisa langsung diakses oleh CPU , tapi harus melalui register.

Mesti diingat! kita tdk bisa mencopy sebuah nilai langsung ke I/O register atau RAM harus melalui registers



sram

Perpindahan data di antara Data memori (GPR-SFR-SRAM) dan Instruksinya antara lain:

– SRAM ke GPR : LDS

– GPR ke SRAM : STS

– SFR ke SRAM : none

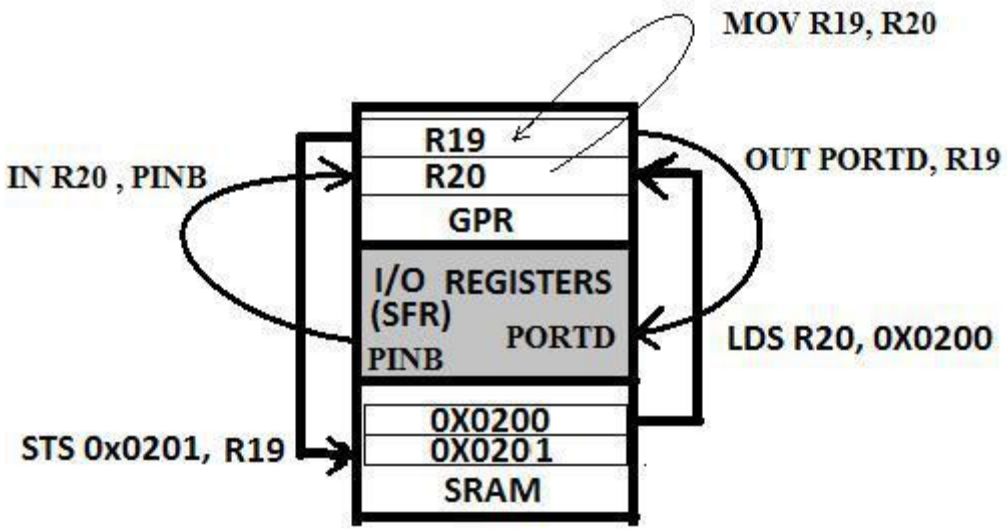
– SRAM ke SFR : none

– SRAM ke SRAM : none

– GPR ke GPR : MOV

– SFR ke GPR : IN

– GPR ke SFR : OUT



Instruksi perpindahan data memori

Perintah untuk mengakses RAM adalah LDS dan STS, contoh :

STS 0x0060, R1
isi dari register R1 di copy ke lokasi 0x0060 di SRAM

LDS R1, 0x0060

isi SRAM alamat 0x0060 di copy ke register.

Contoh penjumlahan 2 bilangan dilokasi memory , step yg akan dilalui sbb :

1. Bilangan pertama dicopy dari RAM ke R3

2. Bilangan ke dua di copy dari RAM ke R2

3 ALU akan menjumlahkan R2 dan R3 ,

4 Hasil disimpan ke R3 dan di copy ke RAM.

EEPROM Data Memory

ATMega8535 memiliki EEPROM sebesar 512 byte untuk menyimpan data
Nonvolatile artinya jika power off data tidak hilang. Lokasinya terpisah dengan system address register.

Register-register khusus untuk mengakses EEPROM yaitu :

1. EEADRH dan EEADRL = register menyimpan alamat EEPROM tujuan

2. EEDR = data yang akan disimpan ke EEPROM di copy ke register ini.

3. EECR = register untuk pengontrolan menulis dan membaca. yang digunakan cuma bit 3~ bit 0 :

• Bit 3– EERIE: EEPROM Ready Interrupt Enable
• Bit 2 – EEMWE: EEPROM Master Write Enable
• Bit 1 – EEWE: EEPROM Write Enable
• Bit 0 – EERE: EEPROM Read Enable

waktu penulisan ke EEPROM lebih lama dari pada ke RAM.

Contoh Code untuk menulis data yg ada di register R16 ke EEPROM:

```
; tunggu penulisan sebelumnya komplit

sbic EECR,EEWE

rjmp EEPROM_write

; simpan alamat EEPROM tujuan (misal ada di R17 dan R18) ke register EEARH (MSB) dan EEARL (LSB)

out EEARH, r18

out EEARL, r17

out EEDR,r16                ; data di r16 di copy ke Data Register EEPROM

sbi EECR,EEMWE              ; Write logical one to EEMWE

; Start eeprom write by setting EEWE

sbi EECR,EEWE
```

Register Status

Status Register adalah register yang memberikan informasi yang dihasilkan dari eksekusi instuksi aritmatika.Informasi ini berguna untuk mencari alternatif alur program sesuai dengan kondisi yang dihadapi.

Bit	7	6	5	4	3	2	1	0
	I	T	H	S	V	N	Z	C
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

Register Status

Bit 7 – I : Global Interrupt Enable

Jika bit Global Interrupt Enable diset, maka fasilitas interupsi dapat dijalankan. Bit ini akan clear ketika ada interrupt

yang dipicu dari hardware, setelah program interrupt dieksekusi, maka bit ini harus di set kembali dengan instruksi SEI.

Bit 6 – T : Bit Copy Storage

Instruksi bit copy BLD dan BST menggunakan bit T sebagai sumber atau tujuan dalam operasi bit.

Bit 5 – H : Half Carry Flag

Bit 4 – S : Sign Bit

Bit S merupakan hasil exlusive or dari Negative Flag N dan Two's Complement Overflow Flag V.

Bit 3 – V : Two's Complement Overflow Flag

Digunakan dalam operasi aritmatika

Bit 2 – N : Negative Flag

Jika operasi aritmatika menghasilkan bilangan negatif, maka bit ini akan set.

Bit 1 – Z : Zero Flag

Jika operasi aritmatika menghaslkan bilangan nol, maka bit ini akan set.

Bahasa Assembler AVR

Bahasa yang dipakai untuk memprogram mikrokontroler AVR adalah bahasa assembly AVR atau bahasa C. Bahasa assembler digunakan karena kita dapat melihat perubahan isi register dan data memory.

Program bahasa assembler terdiri dari 2 bagian yaitu

1. Pengarah */directive*

Antara lain: INCLUDE, EQU,SET,ORG

Contoh Pengarah :

.include "m8535def.inc"

.org 0x0000

2. Instruksi .

Format instruksi : label: mnemonic operand ;comment
Contoh Instruksi MOV R1,R2

Contoh program assembler sederhana

Jalankan Program dibawah ini dengan AVR STudio

.include "m8535def.inc" //file definisi jenis microcontroller

.org 0x0000 //set alamat awal (original)

rjmp main

main: ldi R16,low(RAMEND) //lokasi akhir RAM untuk stack(SP)

out SPL,R16 //LSB

ldi R16,high(RAMEND)

out SPH,R16 //MSB

ulang: ldi R16,0xff

out ddra, R16 //port A sebagai output

cbi PortA,1 //pin 1 portA=0

sbi PortA,1 //pin 1 portA=1

rjmp ulang

Langkah langkah

– Buka AVR Studio IDE

– Buka project baru , pilih type project : AVR Assembler

– Ketik nama project : tugas1

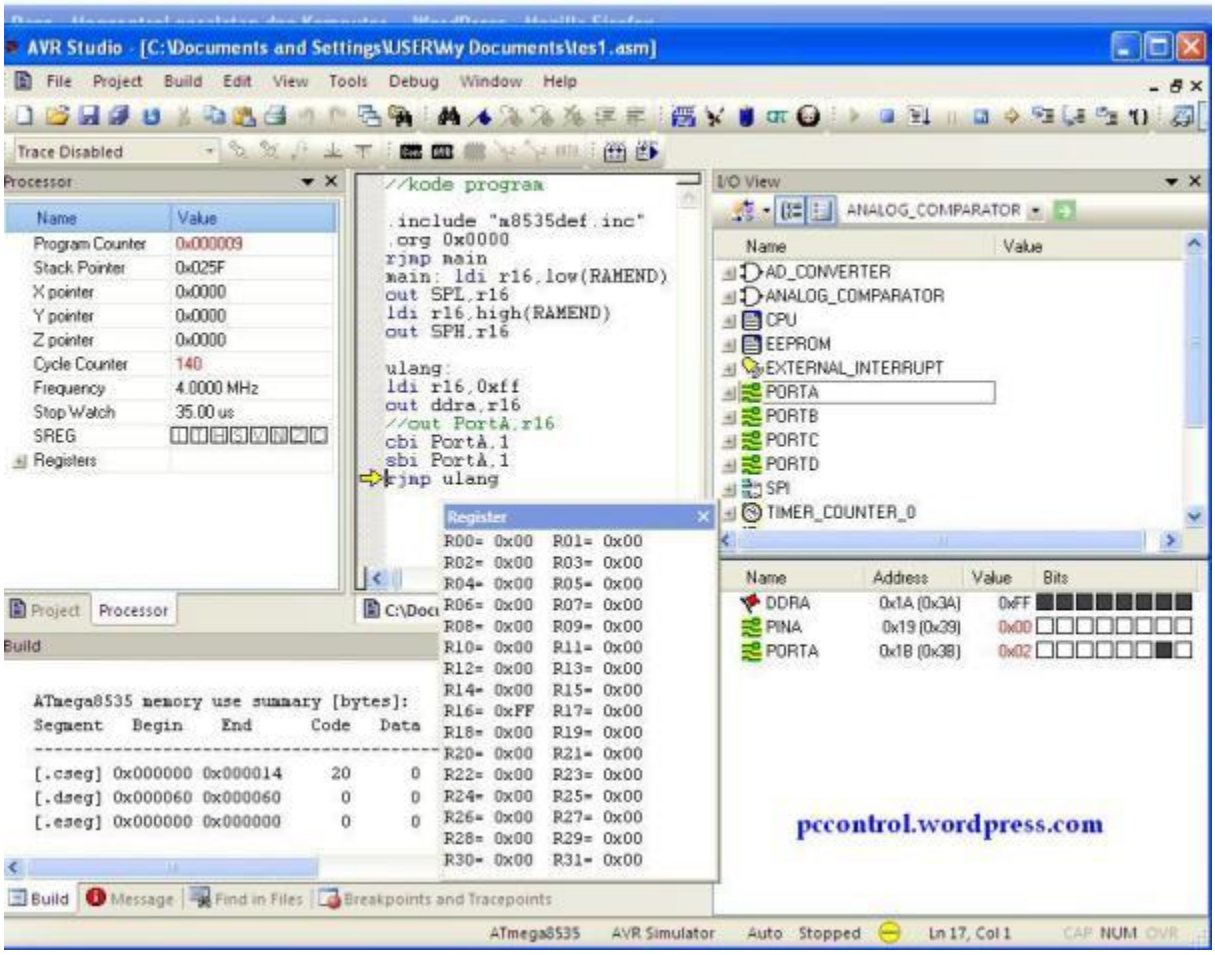
– Pilih Debug Flatform : AVR Simulator

– Pilih Device : ATmega8535 lalu klik finish.

– Ketik program diatas di jendela kode.

– Klik Build and Run untuk menjalankan program.

– Tekan tombol F11 untuk mengeksekusi tiap baris instruksi.



Assembler AVR dgn AVR Studio

Penjelasan Program

.include “m8535def.inc”

Baris ini digunakan untuk mMenentukan jenis mikrokontroler yang digunakan dengan cara memasukkan file definisi device (m8535def.inc) ke dalam program utama.

.org 0x0000

Baris ini digunakan untuk menuliskan awal alamat program, yaitu 0x0000. Hal ini dimaksudkan agar program memory tidak tumpang tindih dengan data memory.

ldi r16,low(RAMEND) //lokasi akhir RAM untuk stack(SP)

out SPL,r16 //LSB

ldi r16,high(RAMEND)

out SPH,r16 //MSB

Empat baris instruksi ini untuk menentukan isi Stack Pointer dengan address terakhir RAM (RAMEND). Untuk ATMega8535 yaitu 0x025F.

Contoh 2 , program penjumlahan isi register 16 dan register 17 ,hasil disimpan di register16.

```
.include "m8535def.inc"
.org 0x00
rjmp main
main: ldi r16,low(RAMEND)
out SPL,r16
ldi r16,high(RAMEND)
out SPH,r16

ldi r16,0x08
ldi r17,0x80
add r16,r17              ; R16 =R16+R17
here: rjmp here
```

beberapa Instruksi Assembler

instruksi transfer data

Instruksi	Arti	contoh
LDI Rd,K RD = K .d=16-31. K = 0-255	Artinya copy data 8 bit ke GPR (hanya R16~R31).	LDI R16, 0xf LDI R31,8 //desimal 8
ADD Rd,Rr	Rd = Rd + Rr	
LDS Rd,K(LoaD direct to data Space)Rd = isi alamat K , d = 0~31 K = 0~FFFF	Copy isi alamat K ke RD	LDS R20,0x1.copy alamat 0x0001 (R1) ke R20.
STS K, Rr(Store direct To data Space)	Mengcopy isi register Rr ke lokasi memory K	1) STS 0x1 ,R10copy isi R10 ke alamat 0x0001 (R1).2) LDI R16,0x55 STS 0x38,R16 // 0x38(PortB) = 0x55
IN Rd,A	isi GPR dgn memori relatif SFR	IN R20,0x16 (PINB=mem adress 0x36, relatif adress 0x16) artinya isi R20 = PINB untuk instruksi IN lebih baik ganti "A" dgn nama jadi IN R20,0x16 sama dengan IN R20 ,PINB
OUT A,Rr	**isi reg i/o dengan Rr**	**Out PORTD,R10**
JMP	spt goto bahasa C	lagi : IN R16, PINB OUT PORTC,R16 JMP lagi
MOV	**Mengcopy data di antara register GPR**	**MOV R10, R20**

Instruksi Operasi Aritmatika

		Contoh :
ADD	Menambahkan isi dua register.	add r15,r14 ; r15=r15+r14
ADC	Menambahkan isi dua register dan isi carry flag	adc r15,r14 ; r15=r15+r14+C
SUB	Mengurangi isi dua register.	sub r19,r14 ; r19=r19-r14
MUL	Mengalikan dua register. Perkalian 8 bit dengan 8 bit menghasilkan bilangan 16 bit yang disimpan di r0untuk byte rendah dan di r1 untuk byte tinggi. Untuk memindahkan bilangan 16 bit antar register registerdigunakan instruksi **movw (copy register word)**	mul r21,r20 ; r1:r0=r21*r20

Instruksi Logika

Instruksi	Arti	Contoh
AND Rm,Rn	Rm=Rm & Rn	AND Rr23,R27
ANDI Rn,kontanta	Rn=Rn & konstanta	ANDI R25,0b11110000
OR	R18 = R18 OR R17	OR R18, R17
ORI	Rn = Rn OR konstanta	ori r15,0xfe
INC	Rn = Rn + 1	INC R16
DEC	Rn = Rn – 1	DEC R16
CLR	Rn = 0	CLR R15 ; R15=0x00
SER	Rn=0xFFh	SER R16 ; r16=0xff

Instruksi I/O

Instruksi	Arti	Contoh
IN	membaca data I/O port ke dalam register	IN R16,PinA
OUT	menulis data register ke I/O port	OUT PortA,R16
LDI(load immediate)	menulis konstanta ke register sebelum konstanta tersebut	LDI R16,0xFF
SBI(set bit in I/O)	membuat logika high pada sebuah bit I/O port	SBI PortB,7
CBI(clear bit in I/O)	membuat logika low pada sebuah bit I/O port	CBI PortB,5
SBIC(skip if bit in I/O is	lompati satu instruksi jika bit	SBIC PortA,3

clear)	I/O port dalam kondisi clear/low	
SBIS(skip if bit in I/O is set)	lompati satu instruksi jika bit I/O port dalam kondisi set/high	SBIS PortB,3

Operasi Percabangan

Instruksi Percabangan

istruksi	**arti**	**contoh**
Sbic (skip if bit in I/O is leared)	Skip jika bit I/O yang diuji clear	**SBIC PINB,0**; Skip if Bit 0 on port B is 0 **RJMP ATarget** ; Jump to the label ATarget
Sbis (skip if bit in I/O is set)	Skip jika bit I/O yang diuji set	
sbrc (skip if bit in register is lear)	Skip jika bit dalam register yang diuji clear	
cp (compare)	Membandingkan isi dua register	cp r16,r18 ;brne lompat ;(menuju lompat jika r16=r18)
cpi (compare with immediate)	Membandingakan isi register dengan konstanta tertentu	cpi r16,5 ; r16=5 ?breq lagi ;(menuju lagi jika r16 = 5
breq (branch if equal)	Lompat ke label tertentu jika suatu hasil perbandingan adalah sama	
brne (branch if not equal)	Lompat ke label tertentu jika suatu hasil perbandingan adalah tidaksama	
rjmp (relative jump)	Lompat ke label tertentu	
rcall (relative call)	Memanggil subrutin.	
ret (return)	Keluar dari sub rutin.	
CPSE(ComPare Skip if Equal)	Compare R1 and R2, skip if equal	**CPSE R1,R2**

==========

Contoh contoh Program

1. **Operasi Percabangan**

.include "m8535def.inc"

.org 0x00

rjmp main

main: ldi r16,low(RAMEND)

out SPL,r16

ldi r16,high(RAMEND)

out SPH,r16

clr r16 ; r16=0x00

naik: inc r16 ; increment r16

cpi r16,5 ; r16=5 ?

breq lagi ; branch to lagi if r16 = 5

rjmp naik ; jump to naik if r16 ≠ 5

lagi: ldi r18,5 ; r18 = 5

dec r16 ; decrement r16

cp r16,r18 ; compare r16 & r18

brne lompat ; branch to lompat if r16=r18

rjmp lagi ; jump to lagi if r16≠r18

lompat: rcall rutin1

rcall rutin2

henti: rjmp henti

rutin1: mov r17,r16

ret

rutin2: mov r19,r18

ret

2. perpindahn data Memori

.include "m8535def.inc"

.org 0x00

```
        rjmp main
main:  ldi r16,low(RAMEND)
out SPL,r16
ldi r16,high(RAMEND)
out SPH,r16
lagi:    LDI r18, 3            ; r18=3
LDI r19, 2            ; r19=2
ADD  r19, r18        ;r19 =r19+r18
STS  0x60, r19      ; copy isi r19 ke lokasi SRAM alamat 0x60
LDS  R20, 0x60    ;copy isi alamat SRAM 0x60 ke reg r20
rjmp lagi
```

3. Input Output Port

```
.include "m8535def.inc"
.org 0x00
rjmp main
main:  ldi r16,0x00
  out ddra,r16          ; PortA as input
  ldi r16,0xff
  out ddrb,r16          ; PortB as output
ulang: in r16,PinA        ;baca PinA simpan di r16
      out PortB,r16    ; kirim isi register r16 ke portB
     rjmp  ulang
```

4. Aritmetika

```
.include "m8535def.inc"
.org 0x00
```

rjmp main

main: ldi r16,low(RAMEND)

out SPL,r16

ldi r16,high(RAMEND)

out SPH,r16

ldi r16,0x80

ldi r17,0x80

add r16,r17

ldi r18,0x02

adc r16,r18

here: rjmp here

5. Operasi Logika

.include "m8535def.inc"

.org 0x00

rjmp main

main:

ldi r16,low(RAMEND)

out SPL,r16

ldi r16,high(RAMEND)

out SPH,r16

ldi r16,0b01110111

ldi r17,0b00001111

and r16,r17

ori r16,0b00001000

clr r16

inc r16

ser r16

dec r16

here:

rjmp here

6. Percabangan

.include "m8535def.inc"

.org 0x00

rjmp main

main: ldi r16,low(RAMEND)

out SPL,r16

ldi r16,high(RAMEND)

out SPH,r16

clr r16 ; r16=0x00

naik: inc r16 ; increment r16

cpi r16,5 ; r16=5 ?

breq lagi ; branch to lagi if r16 = 5

rjmp naik ; jump to naik if r16 ≠ 5

lagi: ldi r18,5 ; r18 = 5

dec r16 ; decrement r16

cp r16,r18 ; compare r16 & r18

brne lompat ; branch to lompat if r16=r18

rjmp lagi ; jump to lagi if r16≠r18

lompat: rcall rutin1

rcall rutin2

henti: rjmp henti

rutin1: mov r17,r16

ret

rutin2: mov r19,r18

ret

7. Interupsi

.include “m8535def.inc”

.org 0x0000

rjmp main

.org 0x0001

rjmp ex_int0

main: ldi r16,low(RAMEND)

out SPL,r16

ldi r16,high(RAMEND)

out SPH,r16

ldi r16,0xff

out ddrd,r16

out PortD,r16

set_int: ldi r17,0b01000000

out GICR,r17

ldi r17,0b00000000

out MCUCR,r17

sei

loop: rjmp loop

```
ex_int0: push r16
in r16,SREG
push r16
ldi r17,0xff
out ddra,r17
out PortA,r17
pop r16
out SREG,r16
pop r16
reti
```

sumber:

-AVR Ali Mazidi

-Modul AVR Electrical Engineering Batam Polytechnic

– Datasheet AVR Atmega8535