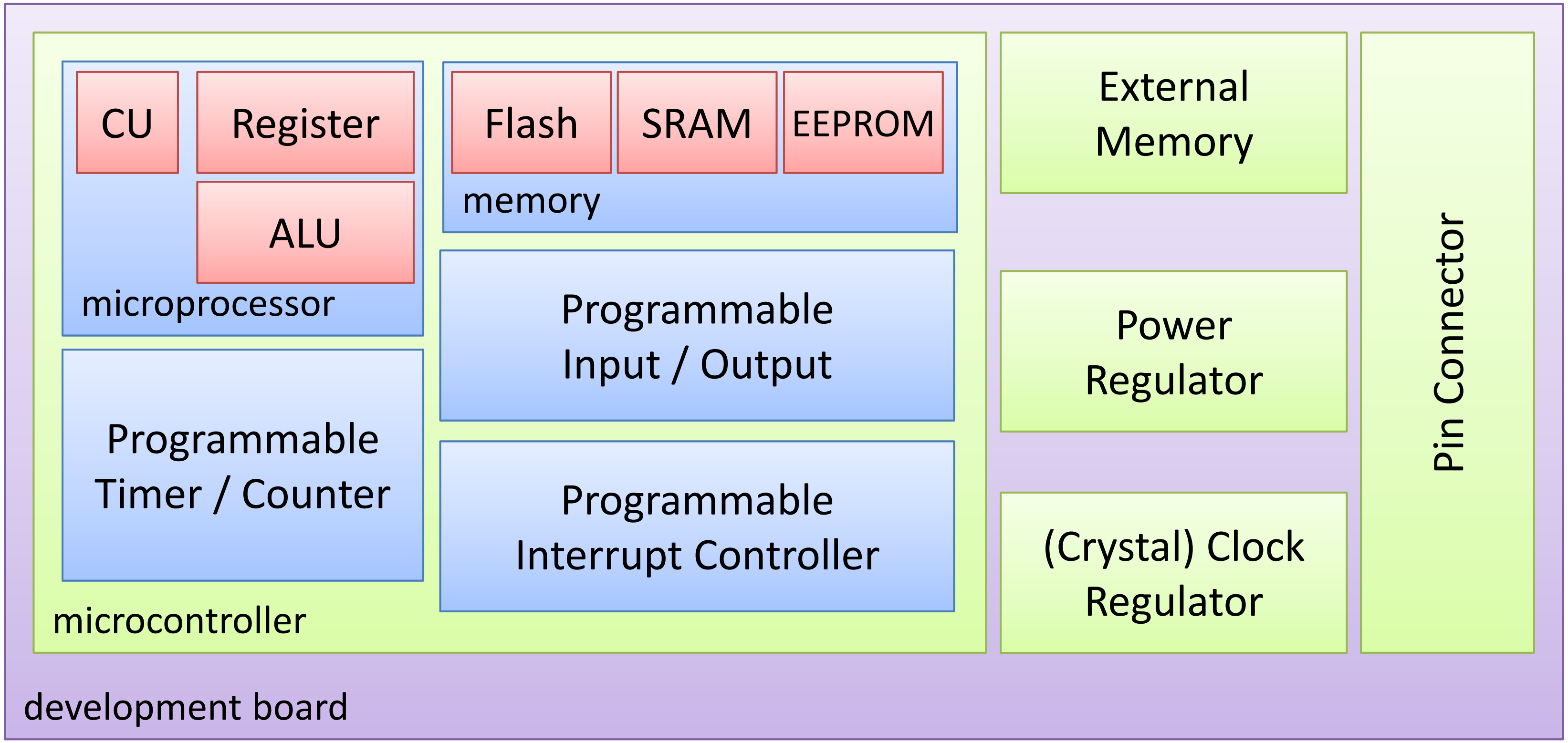
Programming Microcontroller Assembly and C

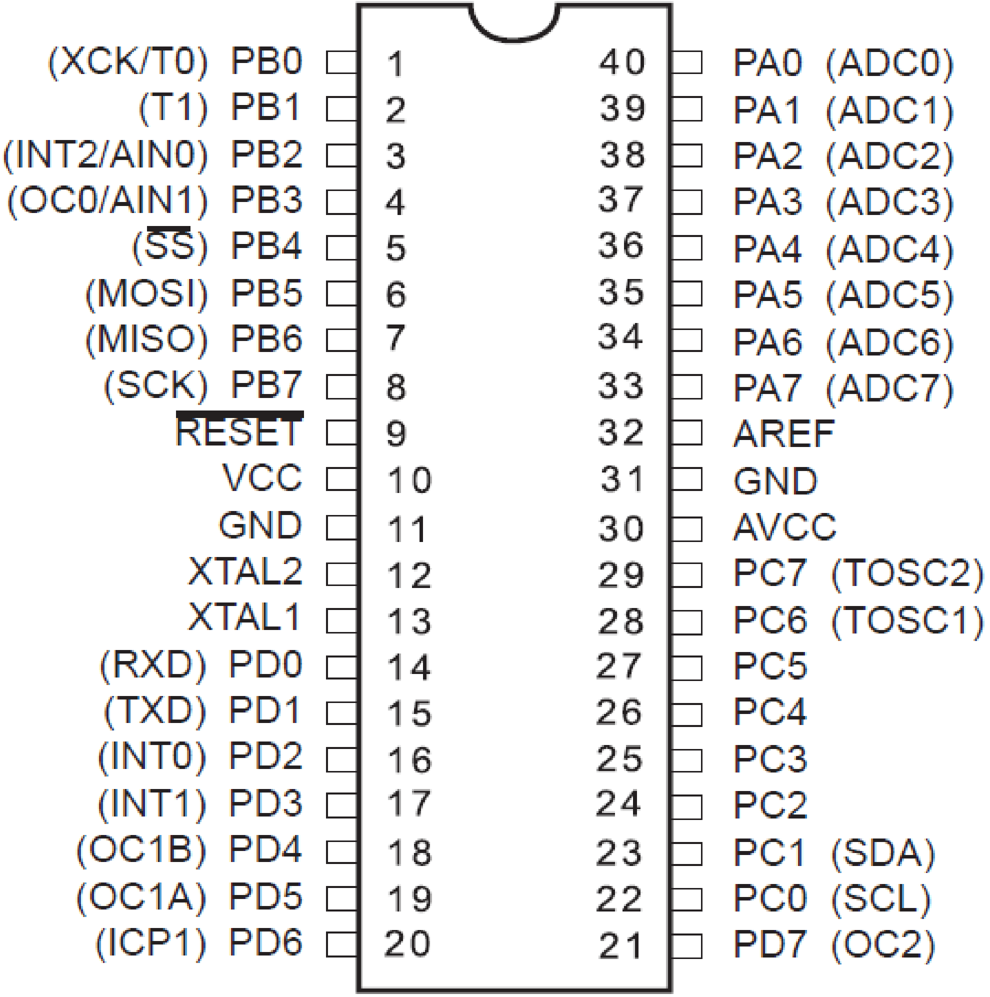
Course Number	: TTH2D3
CLO	: 2
Week	: 5-7

CLO#2 Student have the knowledge to create basic programming for microcontroller
[C3] Understand how to program in Assembly
[C3] Understand how to program a microcontroller using C
[C3] Understand how to store the program in microcontroller

What is the difference between μP, μC, and Development Board?

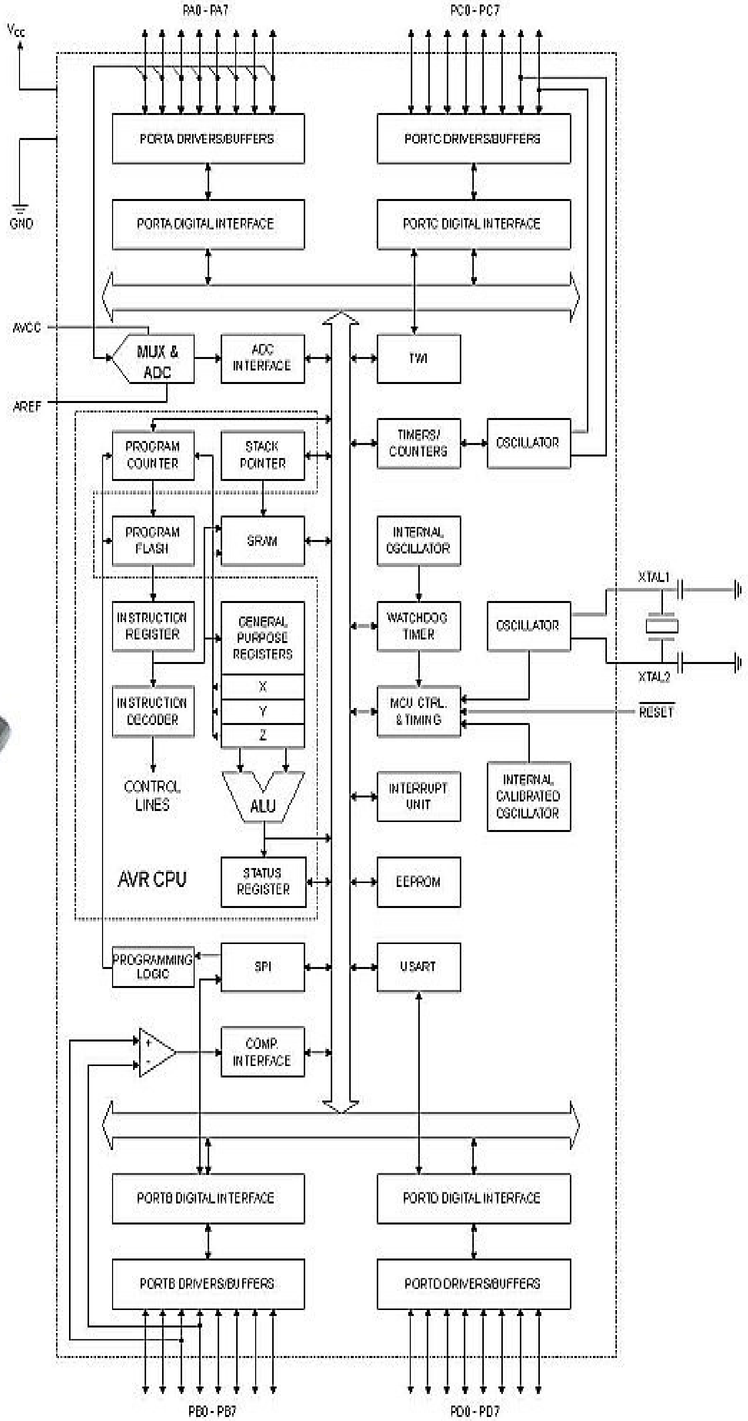


MICROCONTROLLER (μC) AVR ATMEGA 8535



ATMega 8535 Architecture

- 4 I/O port (4x8) Port A, B, C and D
- ADC (Analog to Digital Converter)
- 3 Timer/Counter
- 32 register
- 512 byte SRAM
- 8kb Flash memory
- Internal and external interrupt
- SPI interface port to download program into flash
- 512 byte EEPROM
- Analog comparator interface
- USART port for serial communication



Prerequisite for Programming a Microcontroller

- Before you start to program a μC, you need to understand 2 things:
  1. The Instruction Set of the μC
  2. The hardware capability of the μC
- Most μC comes with an IDE (Integrated Development Environment) to build and run your program
- For Atmel μC, you may choose either Assembly or C for your programming language

Instruction Set

ATMega 32

What is Instruction Set?

- Instruction is like a command to be executed by Control Unit within the μP
- Instruction is usually grouped by 4 types:
  - Data transfer (ex. MOV)
  - Arithmetic and Logic (ex. ADD)
  - Control Transfer (ex. JMP)
  - Miscellaneous (ex. NOP)
- Every μP has their own instruction
- Instruction set is the list of instruction understood only by a specific μP

Basic Instruction (1/2)

- LDI (Load Immediate): writes a constant into a register
  ex. LDI R16,0xFF (writes FF_H into register 16)
- OUT: writes data from register to a specific I/O port
  ex. OUT DDRA,R16 (writes data from register 16 to port A)
- IN: reads data from a specific I/O port into a register
  ex. IN R16, PORTA (reads data from port A and stores it to register 16)
- SBI (Set bit in I/O): set (put into High Voltage) a bit in I/O port
  ex. SBI PORTA,0 (set bit 0 in port A to High Voltage)
- CBI (Clear bit in I/O): unset (put into Low Voltage) a bit in I/O port
  ex. CBI PORTA,1 (set bit 1 in port A to Low Voltage)

Basic Instruction 2/2

- SBIS (Skip if bit in I/O is set): skip 1 instruction below if a specific bit in I/O port is set (High Voltage)
  ex. SBIS PORTA,2

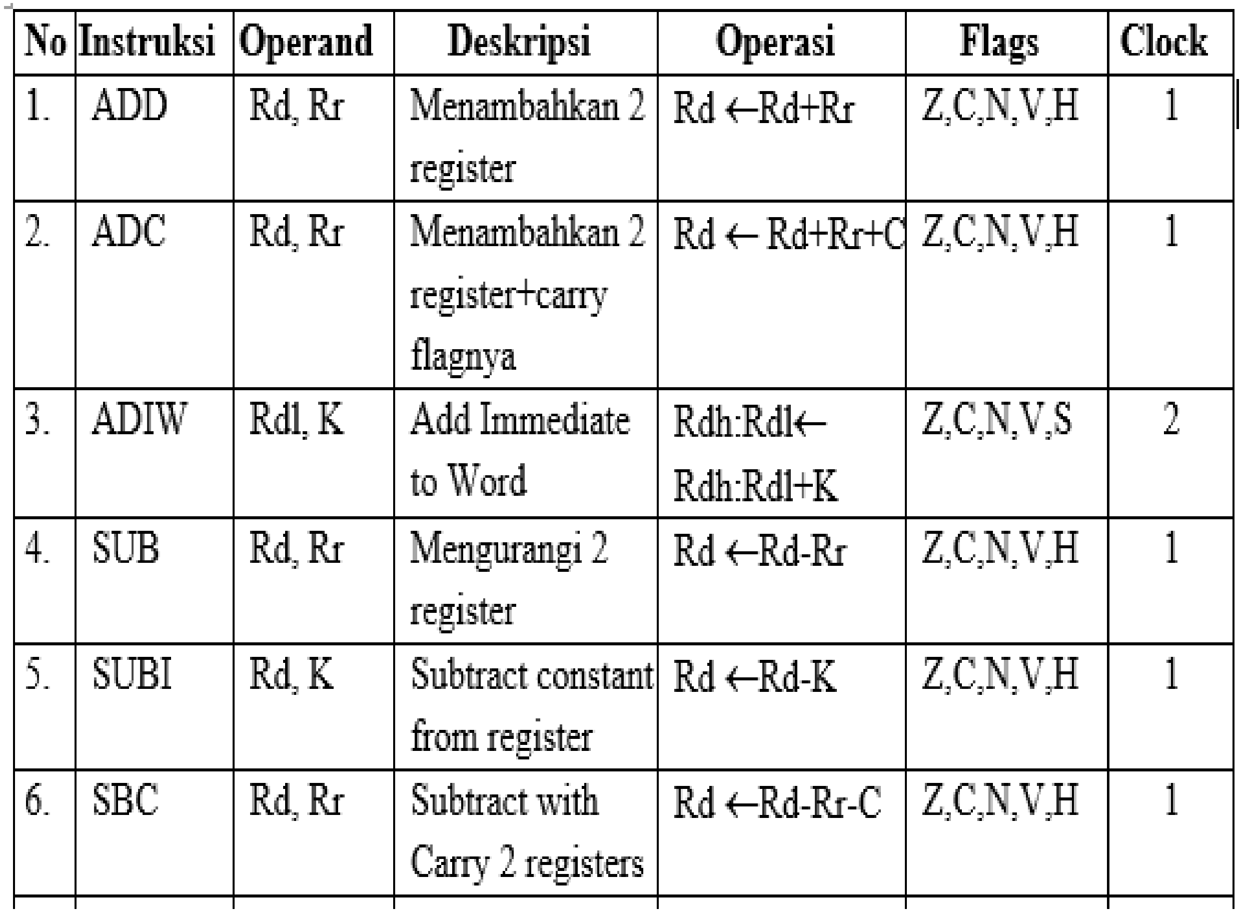
  ```
  RJMP RPT
  ```

  “RJMP RPT” will be skipped if bit 2 in port A is High Voltage
- SBIC (Skip if bit in I/O is cleared): skip 1 instruction below if a specific bit in I/O port is unset (Low Voltage)
  ex. SBIC PORTA,2

  ```
  RJMP RPT
  ```

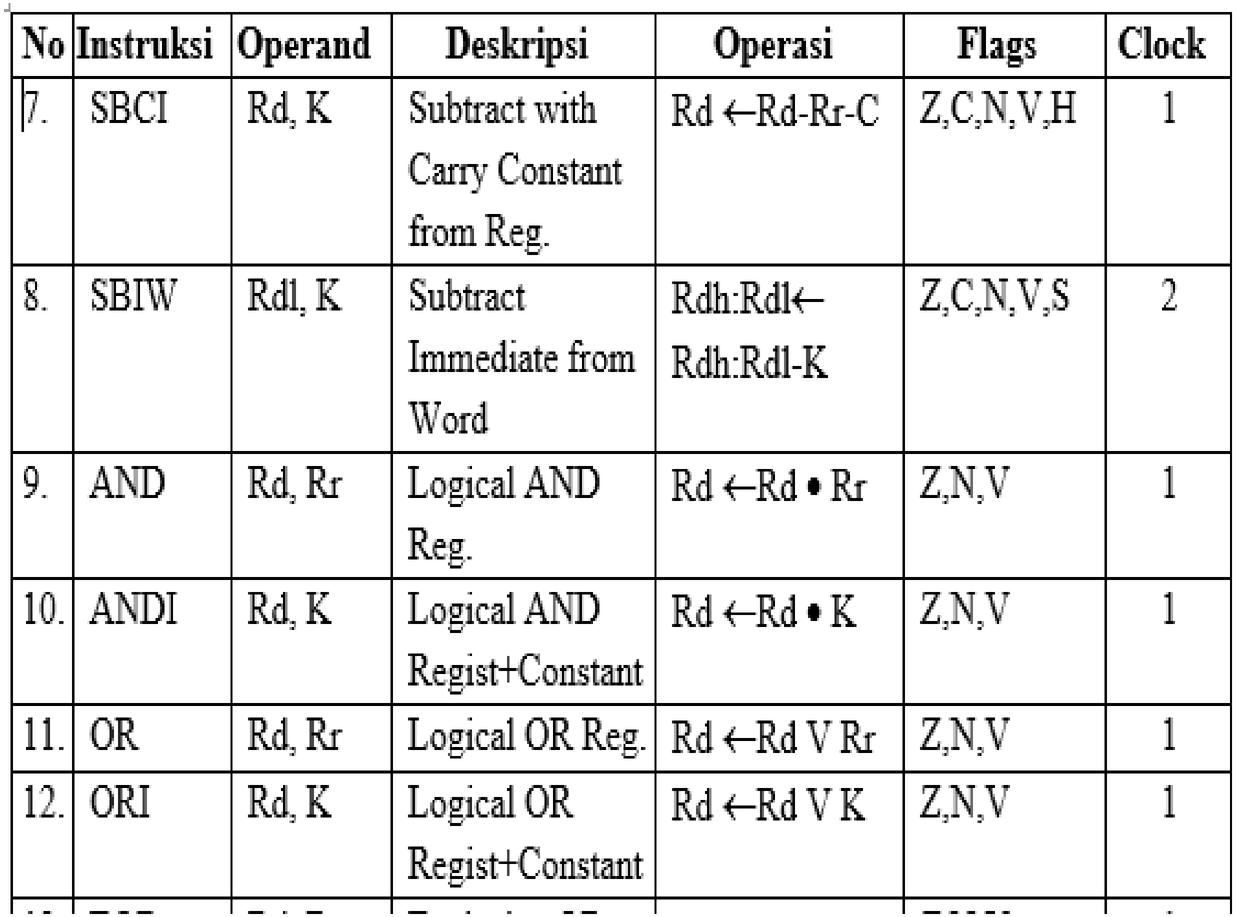
  “RJMP RPT” will be skipped if bit 2 in port A is Low Voltage

Arithmetic Instruction (1/4)



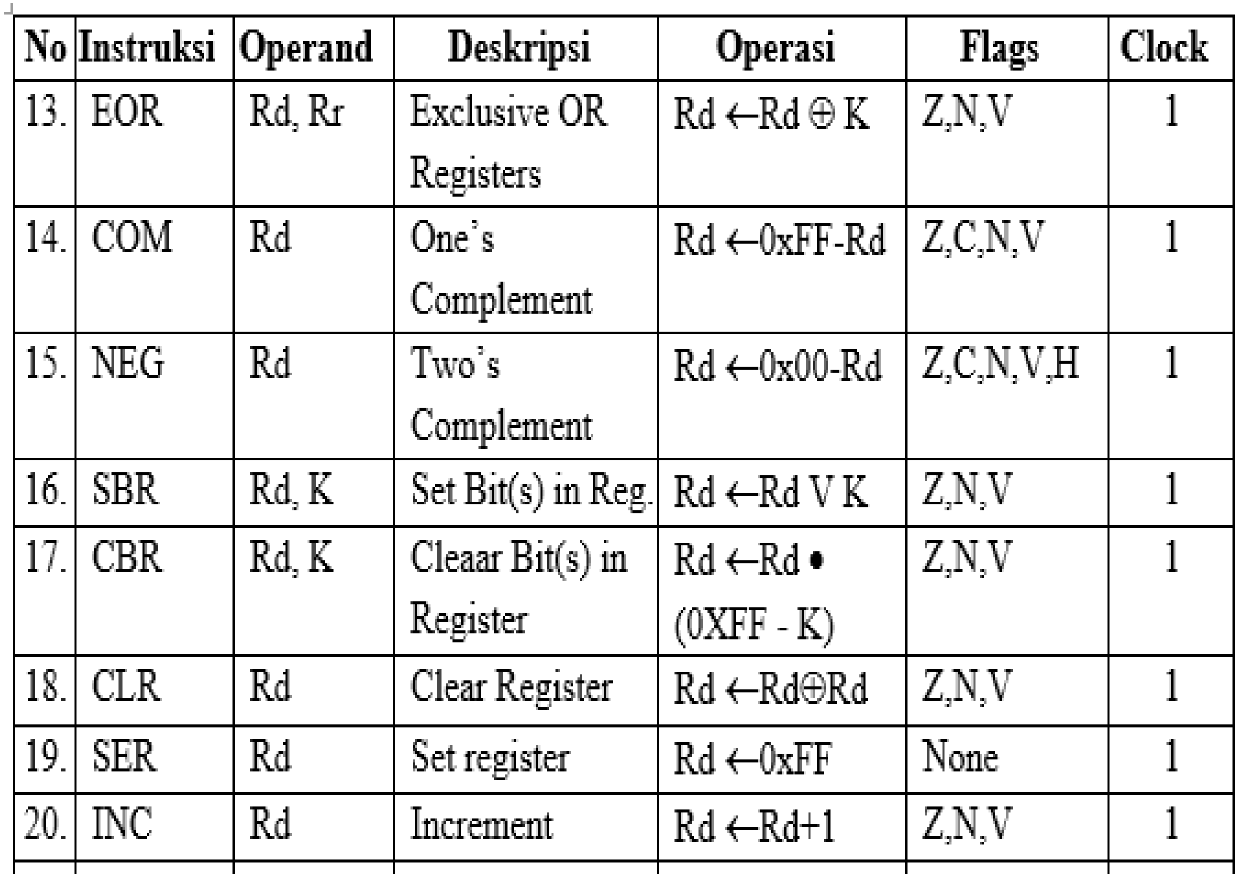
No	Instruksi	Operand	Deskripsi	Operasi	Flags	Clock
1.	ADD	Rd, Rr	Menambahkan 2 register	Rd ←Rd+Rr	Z,C,N,V,H	1
2.	ADC	Rd, Rr	Menambahkan 2 register+carry flagnya	Rd ← Rd+Rr+C	Z,C,N,V,H	1
3.	ADIW	Rdl, K	Add Immediate to Word	Rdh:Rdl← Rdh:Rdl+K	Z,C,N,V,S	2
4.	SUB	Rd, Rr	Mengurangi 2 register	Rd ←Rd-Rr	Z,C,N,V,H	1
5.	SUBI	Rd, K	Subtract constant from register	Rd ←Rd-K	Z,C,N,V,H	1
6.	SBC	Rd, Rr	Subtract with Carry 2 registers	Rd ←Rd-Rr-C	Z,C,N,V,H	1

Arithmetic Instruction (2/4)



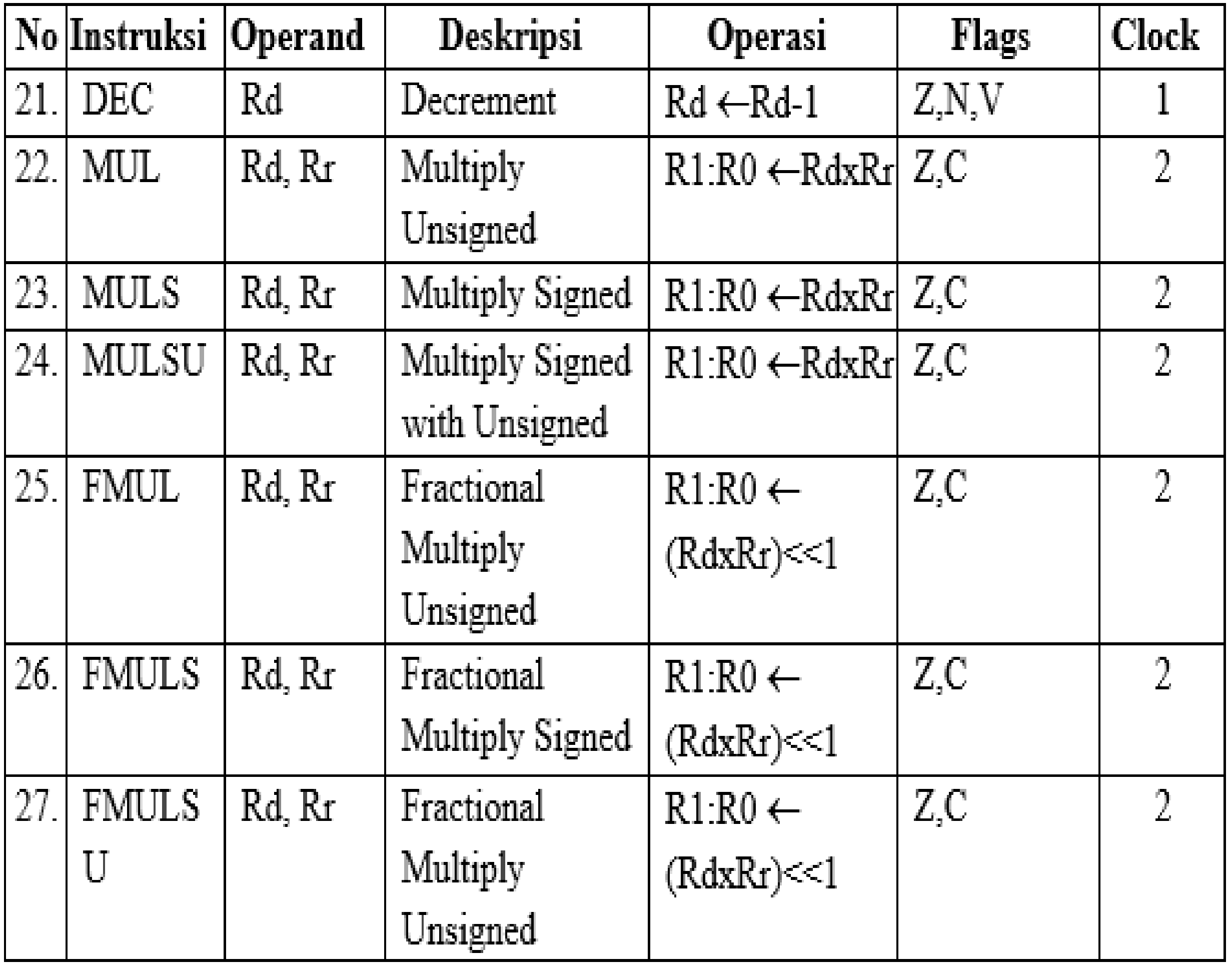
No	Instruksi	Operand	Deskripsi	Operasi	Flags	Clock
7.	SBCI	Rd, K	Subtract with Carry Constant from Reg.	Rd ←Rd-Rr-C	Z,C,N,V,H	1
8.	SBIW	Rdl, K	Subtract Immediate from Word	Rdh:Rdl← Rdh:Rdl-K	Z,C,N,V,S	2
9.	AND	Rd, Rr	Logical AND Reg.	Rd ←Rd • Rr	Z,N,V	1
10.	ANDI	Rd, K	Logical AND Regist+Constant	Rd ←Rd • K	Z,N,V	1
11.	OR	Rd, Rr	Logical OR Reg.	Rd ←Rd V Rr	Z,N,V	1
12.	ORI	Rd, K	Logical OR Regist+Constant	Rd ←Rd V K	Z,N,V	1

Arithmetic Instruction (3/4)



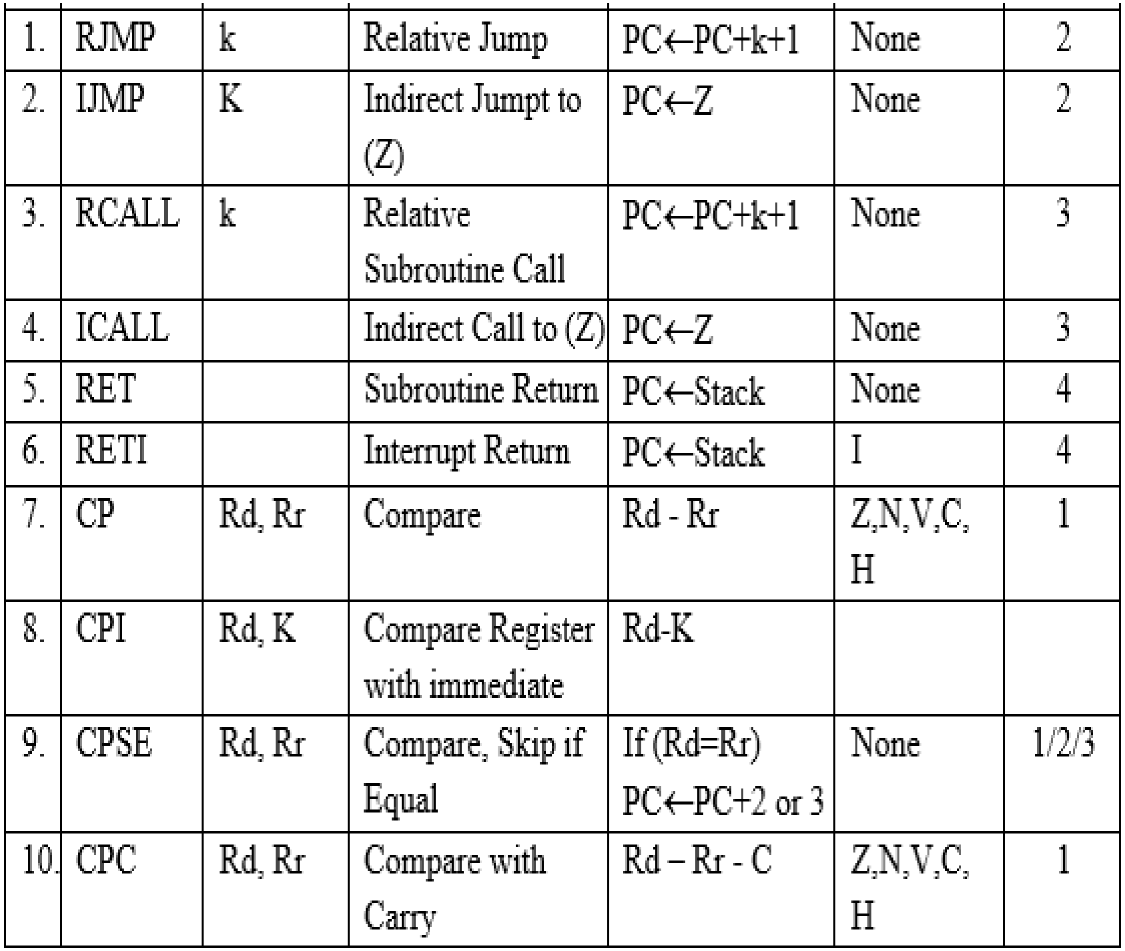
No	Instruksi	Operand	Deskripsi	Operasi	Flags	Clock
13.	EOR	Rd, Rr	Exclusive OR Registers	Rd ←Rd ⊕ K	Z,N,V	1
14.	COM	Rd	One's Complement	Rd ←0xFF-Rd	Z,C,N,V	1
15.	NEG	Rd	Two's Complement	Rd ←0x00-Rd	Z,C,N,V,H	1
16.	SBR	Rd, K	Set Bit(s) in Reg.	Rd ←Rd V K	Z,N,V	1
17.	CBR	Rd, K	Cleaar Bit(s) in Register	Rd ←Rd • (0XFF - K)	Z,N,V	1
18.	CLR	Rd	Clear Register	Rd ←Rd⊕Rd	Z,N,V	1
19.	SER	Rd	Set register	Rd ←0xFF	None	1
20.	INC	Rd	Increment	Rd ←Rd+1	Z,N,V	1

Arithmetic Instruction (4/4)



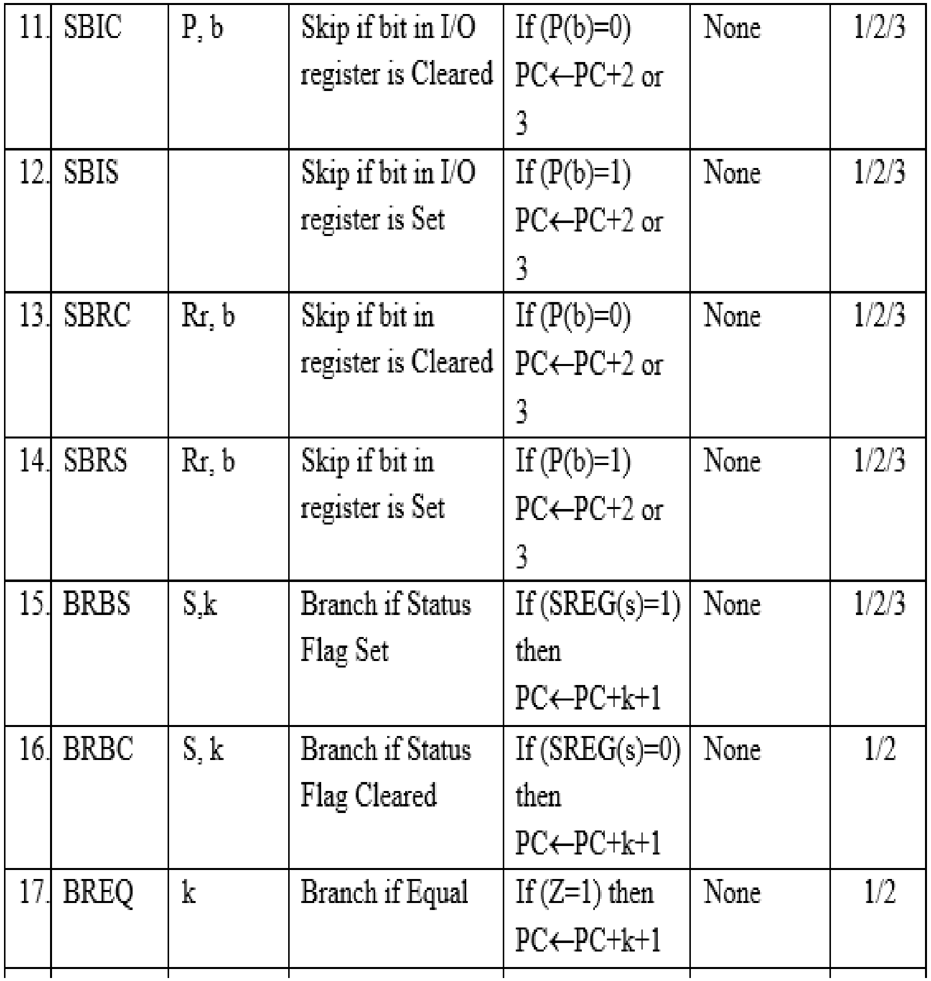
No	Instruksi	Operand	Deskripsi	Operasi	Flags	Clock
21.	DEC	Rd	Decrement	Rd ←Rd-1	Z,N,V	1
22.	MUL	Rd, Rr	Multiply Unsigned	R1:R0 ←RdxRr	Z,C	2
23.	MULS	Rd, Rr	Multiply Signed	R1:R0 ←RdxRr	Z,C	2
24.	MULSU	Rd, Rr	Multiply Signed with Unsigned	R1:R0 ←RdxRr	Z,C	2
25.	FMUL	Rd, Rr	Fractional Multiply Unsigned	R1:R0 ← (RdxRr)<<1	Z,C	2
26.	FMULS	Rd, Rr	Fractional Multiply Signed	R1:R0 ← (RdxRr)<<1	Z,C	2
27.	FMULSU	Rd, Rr	Fractional Multiply Unsigned	R1:R0 ← (RdxRr)<<1	Z,C	2

Branch Instruction (1/4)



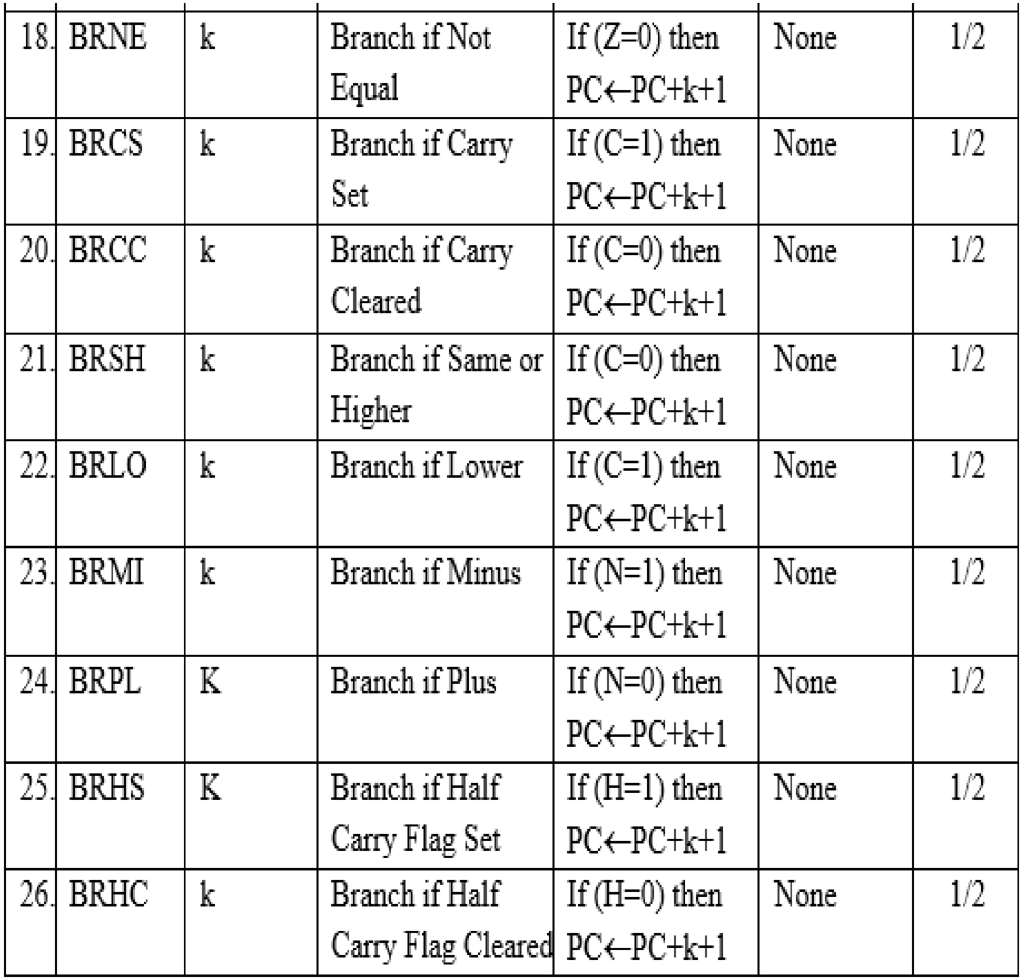
1.	RJMP	k	Relative Jump	PC←PC+k+1	None	2
2.	IJMP	K	Indirect Jumpt to (Z)	PC←Z	None	2
3.	RCALL	k	Relative Subroutine Call	PC←PC+k+1	None	3
4.	ICALL		Indirect Call to (Z)	PC←Z	None	3
5.	RET		Subroutine Return	PC←Stack	None	4
6.	RETI		Interrupt Return	PC←Stack	I	4
7.	CP	Rd, Rr	Compare	Rd - Rr	Z,N,V,C, H	1
8.	CPI	Rd, K	Compare Register with immediate	Rd-K		
9.	CPSE	Rd, Rr	Compare, Skip if Equal	If (Rd=Rr) PC←PC+2 or 3	None	1/2/3
10.	CPC	Rd, Rr	Compare with Carry	Rd – Rr - C	Z,N,V,C, H	1

Branch Instruction (2/4)



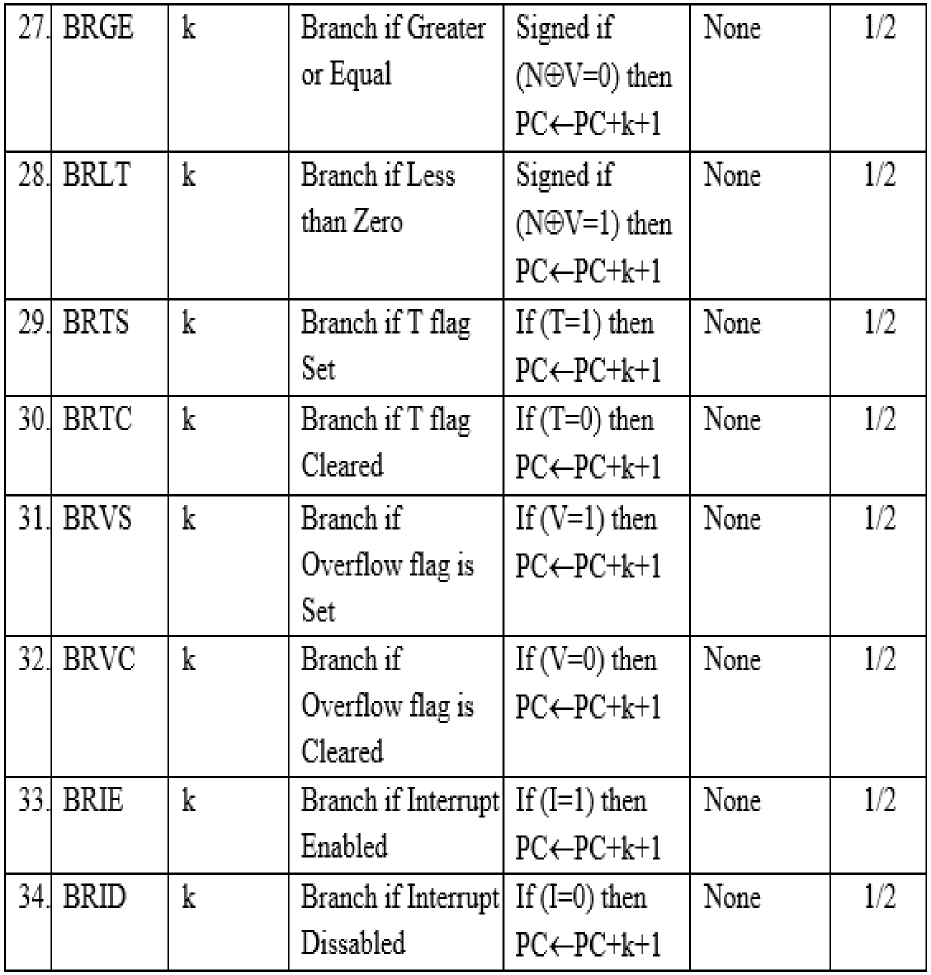
11.	SBIC	P, b	Skip if bit in I/O register is Cleared	If (P(b)=0) PC←PC+2 or 3	None	1/2/3
12.	SBIS		Skip if bit in I/O register is Set	If (P(b)=1) PC←PC+2 or 3	None	1/2/3
13.	SBRC	Rr, b	Skip if bit in register is Cleared	If (P(b)=0) PC←PC+2 or 3	None	1/2/3
14.	SBRS	Rr, b	Skip if bit in register is Set	If (P(b)=1) PC←PC+2 or 3	None	1/2/3
15.	BRBS	S,k	Branch if Status Flag Set	If (SREG(s)=1) then PC←PC+k+1	None	1/2/3
16.	BRBC	S, k	Branch if Status Flag Cleared	If (SREG(s)=0) then PC←PC+k+1	None	1/2
17.	BREQ	k	Branch if Equal	If (Z=1) then PC←PC+k+1	None	1/2

Branch Instruction (3/4)



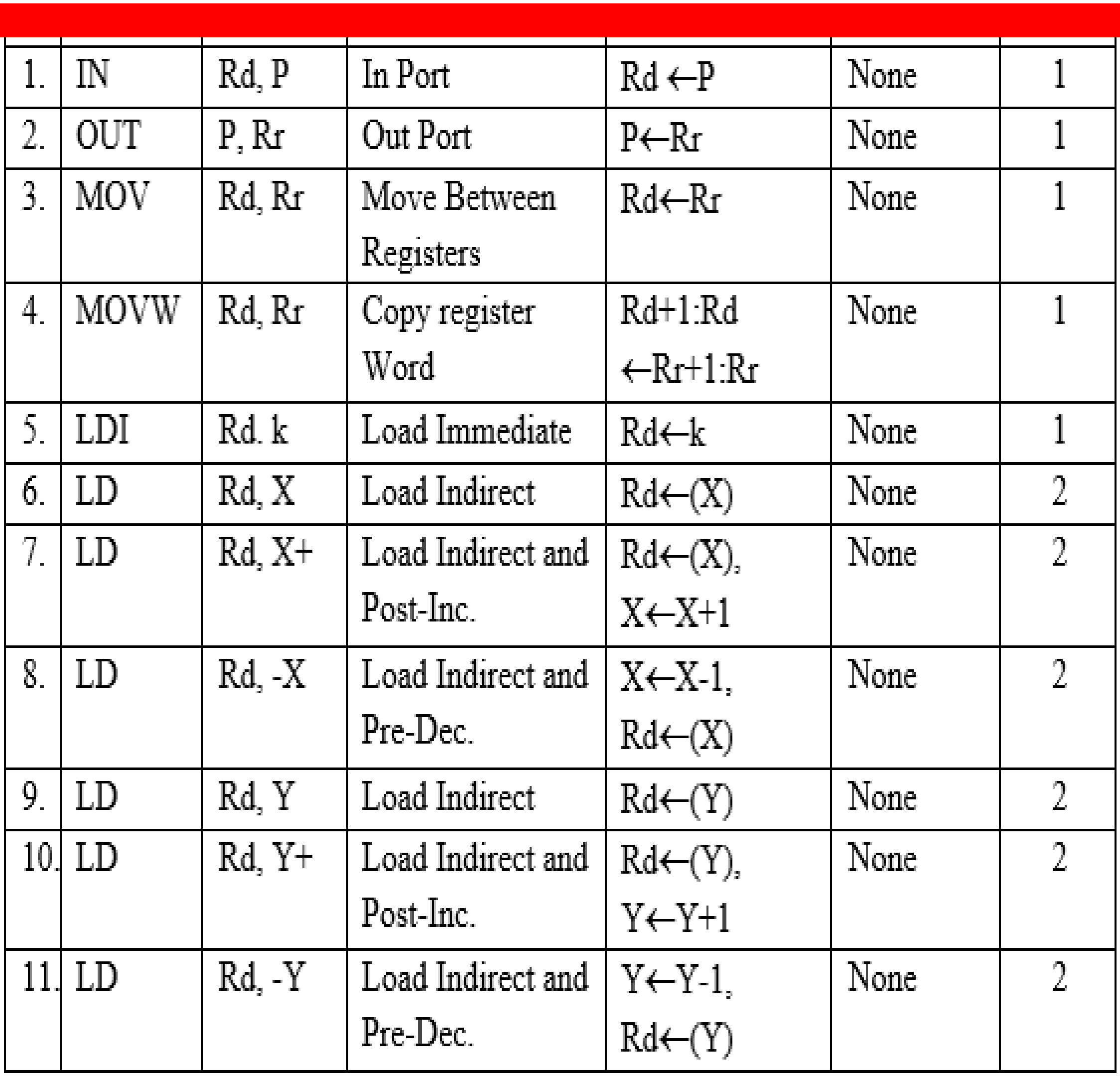
18.	BRNE	k	Branch if Not Equal	If (Z=0) then PC←PC+k+1	None	1/2
19.	BRCS	k	Branch if Carry Set	If (C=1) then PC←PC+k+1	None	1/2
20.	BRCC	k	Branch if Carry Cleared	If (C=0) then PC←PC+k+1	None	1/2
21.	BRSH	k	Branch if Same or Higher	If (C=0) then PC←PC+k+1	None	1/2
22.	BRLO	k	Branch if Lower	If (C=1) then PC←PC+k+1	None	1/2
23.	BRMI	k	Branch if Minus	If (N=1) then PC←PC+k+1	None	1/2
24.	BRPL	K	Branch if Plus	If (N=0) then PC←PC+k+1	None	1/2
25.	BRHS	K	Branch if Half Carry Flag Set	If (H=1) then PC←PC+k+1	None	1/2
26.	BRHC	k	Branch if Half Carry Flag Cleared	If (H=0) then PC←PC+k+1	None	1/2

Branch Instruction (4/4)



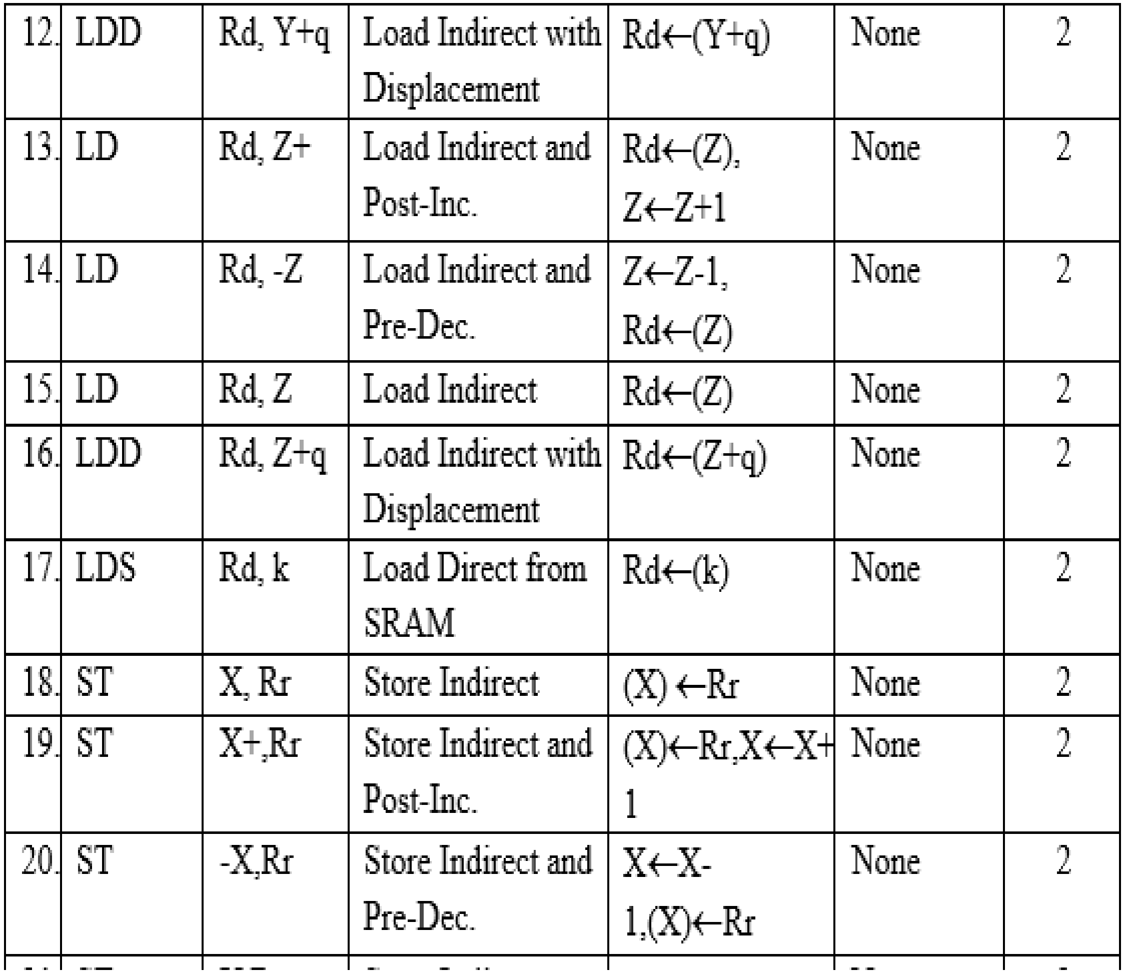
27.	BRGE	k	Branch if Greater or Equal	Signed if (N⊕V=0) then PC←PC+k+1	None	1/2
28.	BRLT	k	Branch if Less than Zero	Signed if (N⊕V=1) then PC←PC+k+1	None	1/2
29.	BRTS	k	Branch if T flag Set	If (T=1) then PC←PC+k+1	None	1/2
30.	BRTC	k	Branch if T flag Cleared	If (T=0) then PC←PC+k+1	None	1/2
31.	BRVS	k	Branch if Overflow flag is Set	If (V=1) then PC←PC+k+1	None	1/2
32.	BRVC	k	Branch if Overflow flag is Cleared	If (V=0) then PC←PC+k+1	None	1/2
33.	BRIE	k	Branch if Interrupt Enabled	If (I=1) then PC←PC+k+1	None	1/2
34.	BRID	k	Branch if Interrupt Dissabled	If (I=0) then PC←PC+k+1	None	1/2

Data Transfer Instruction (1/4)



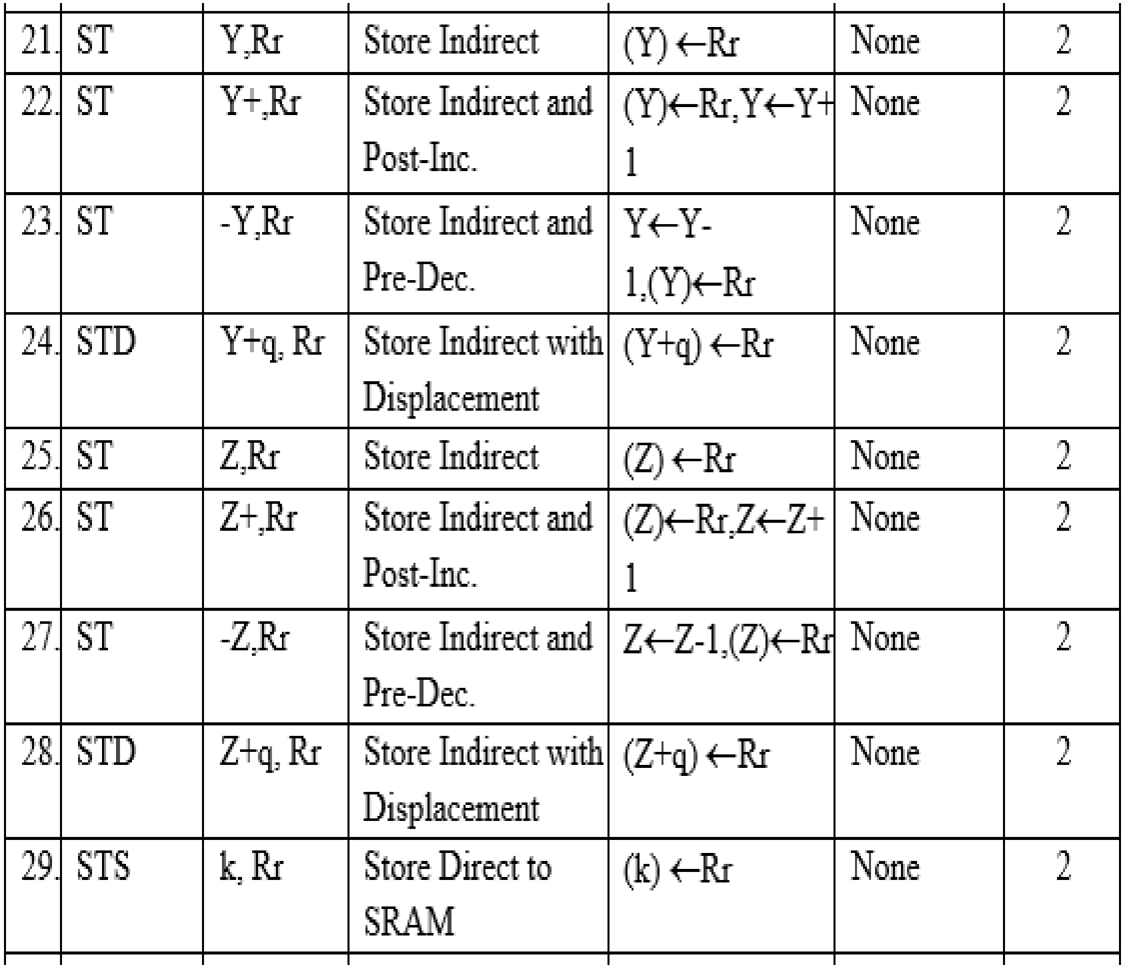
1.	IN	Rd, P	In Port	Rd ←P	None	1
2.	OUT	P, Rr	Out Port	P←Rr	None	1
3.	MOV	Rd, Rr	Move Between Registers	Rd←Rr	None	1
4.	MOVW	Rd, Rr	Copy register Word	Rd+1:Rd ←Rr+1:Rr	None	1
5.	LDI	Rd. k	Load Immediate	Rd←k	None	1
6.	LD	Rd, X	Load Indirect	Rd←(X)	None	2
7.	LD	Rd, X+	Load Indirect and Post-Inc.	Rd←(X), X←X+1	None	2
8.	LD	Rd, -X	Load Indirect and Pre-Dec.	X←X-1, Rd←(X)	None	2
9.	LD	Rd, Y	Load Indirect	Rd←(Y)	None	2
10.	LD	Rd, Y+	Load Indirect and Post-Inc.	Rd←(Y), Y←Y+1	None	2
11.	LD	Rd, -Y	Load Indirect and Pre-Dec.	Y←Y-1, Rd←(Y)	None	2

Data Transfer Instruction (2/4)



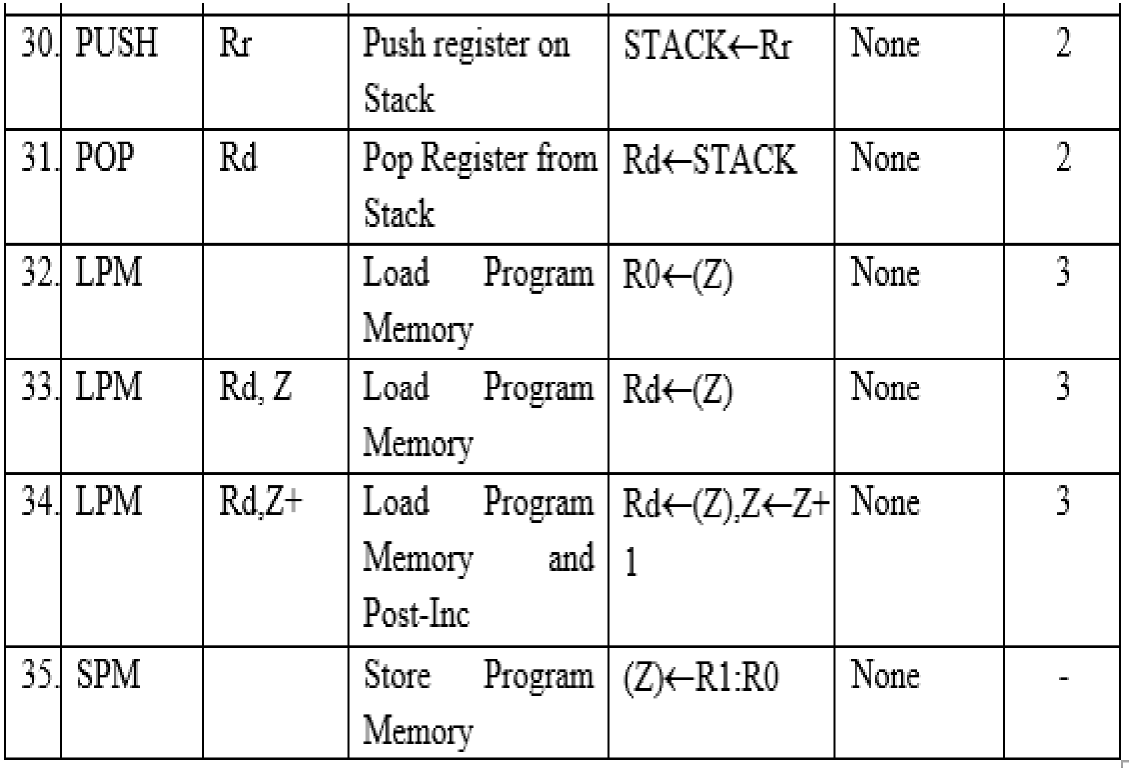
12.	LDD	Rd, Y+q	Load Indirect with Displacement	Rd←(Y+q)	None	2
13.	LD	Rd, Z+	Load Indirect and Post-Inc.	Rd←(Z), Z←Z+1	None	2
14.	LD	Rd, -Z	Load Indirect and Pre-Dec.	Z←Z-1, Rd←(Z)	None	2
15.	LD	Rd, Z	Load Indirect	Rd←(Z)	None	2
16.	LDD	Rd, Z+q	Load Indirect with Displacement	Rd←(Z+q)	None	2
17.	LDS	Rd, k	Load Direct from SRAM	Rd←(k)	None	2
18.	ST	X, Rr	Store Indirect	(X) ←Rr	None	2
19.	ST	X+,Rr	Store Indirect and Post-Inc.	(X)←Rr,X←X+1	None	2
20.	ST	-X,Rr	Store Indirect and Pre-Dec.	X←X-1,(X)←Rr	None	2

Data Transfer Instruction (3/4)



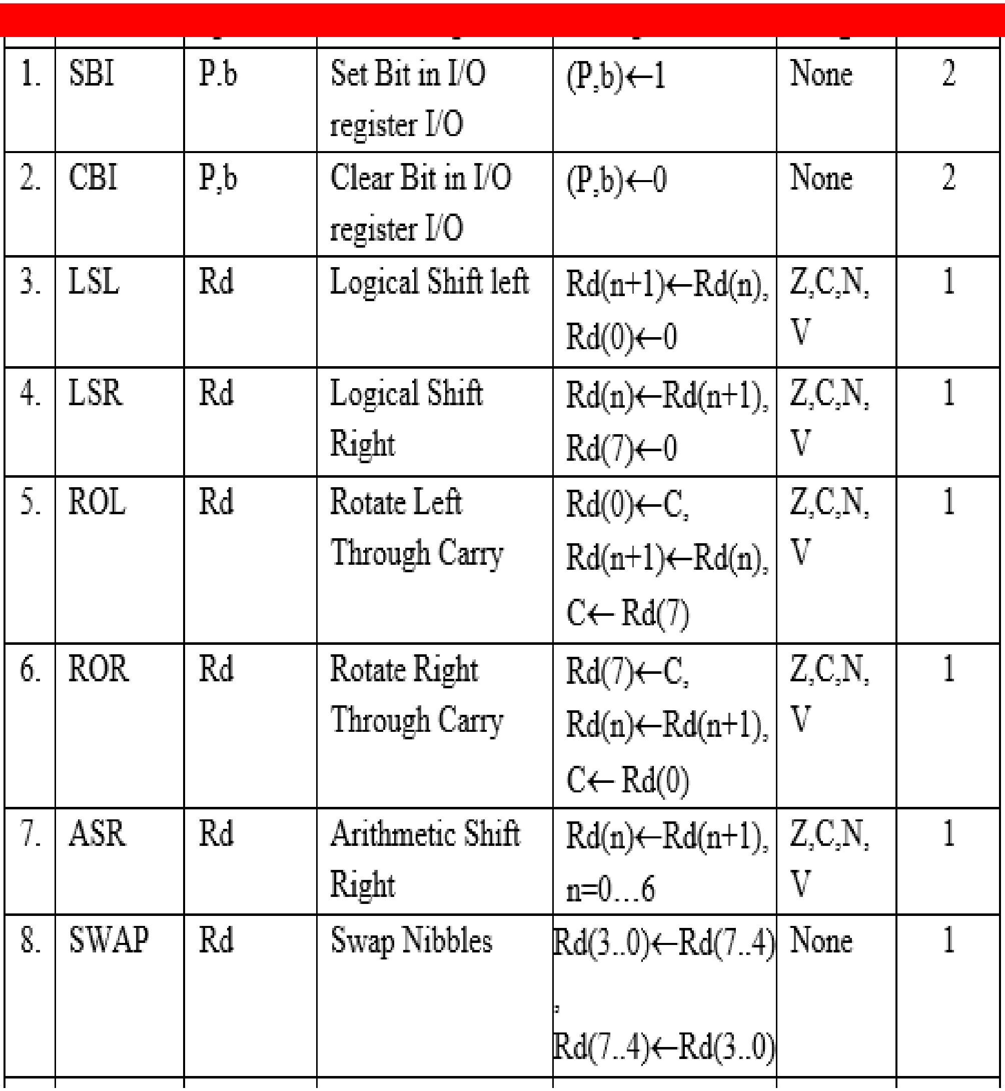
21.	ST	Y,Rr	Store Indirect	(Y) ←Rr	None	2
22.	ST	Y+,Rr	Store Indirect and Post-Inc.	(Y)←Rr,Y←Y+1	None	2
23.	ST	-Y,Rr	Store Indirect and Pre-Dec.	Y←Y-1,(Y)←Rr	None	2
24.	STD	Y+q, Rr	Store Indirect with Displacement	(Y+q) ←Rr	None	2
25.	ST	Z,Rr	Store Indirect	(Z) ←Rr	None	2
26.	ST	Z+,Rr	Store Indirect and Post-Inc.	(Z)←Rr,Z←Z+1	None	2
27.	ST	-Z,Rr	Store Indirect and Pre-Dec.	Z←Z-1,(Z)←Rr	None	2
28.	STD	Z+q, Rr	Store Indirect with Displacement	(Z+q) ←Rr	None	2
29.	STS	k, Rr	Store Direct to SRAM	(k) ←Rr	None	2

Data Transfer Instruction (4/4)



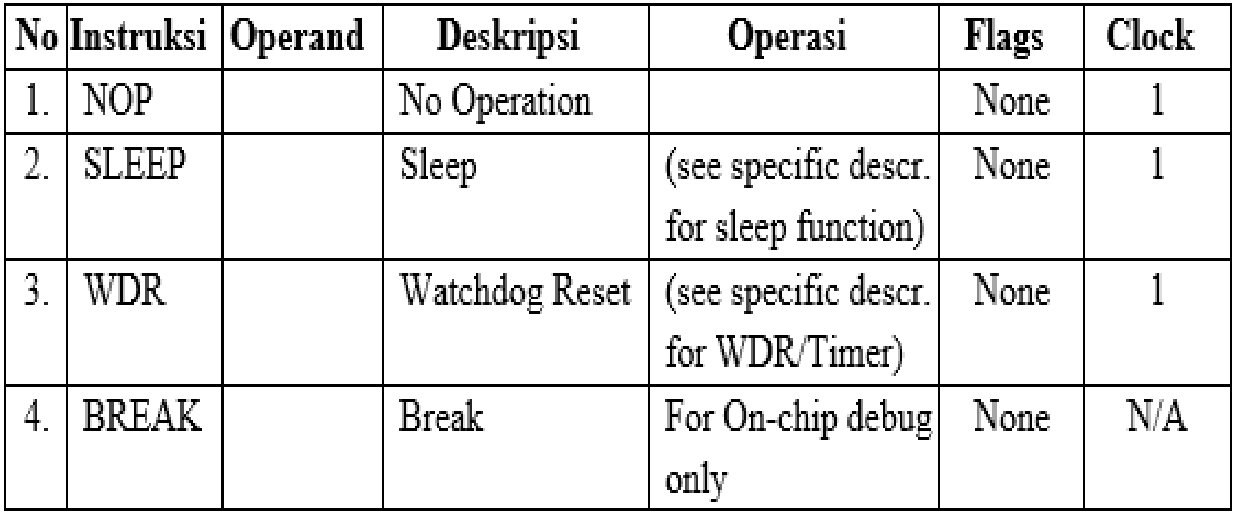
30.	PUSH	Rr	Push register on Stack	STACK←Rr	None	2
31.	POP	Rd	Pop Register from Stack	Rd←STACK	None	2
32.	LPM		Load Program Memory	R0←(Z)	None	3
33.	LPM	Rd, Z	Load Program Memory	Rd←(Z)	None	3
34.	LPM	Rd,Z+	Load Program Memory and Post-Inc	Rd←(Z),Z←Z+1	None	3
35.	SPM		Store Program Memory	(Z)←R1:R0	None	-

Bit Test Instruction



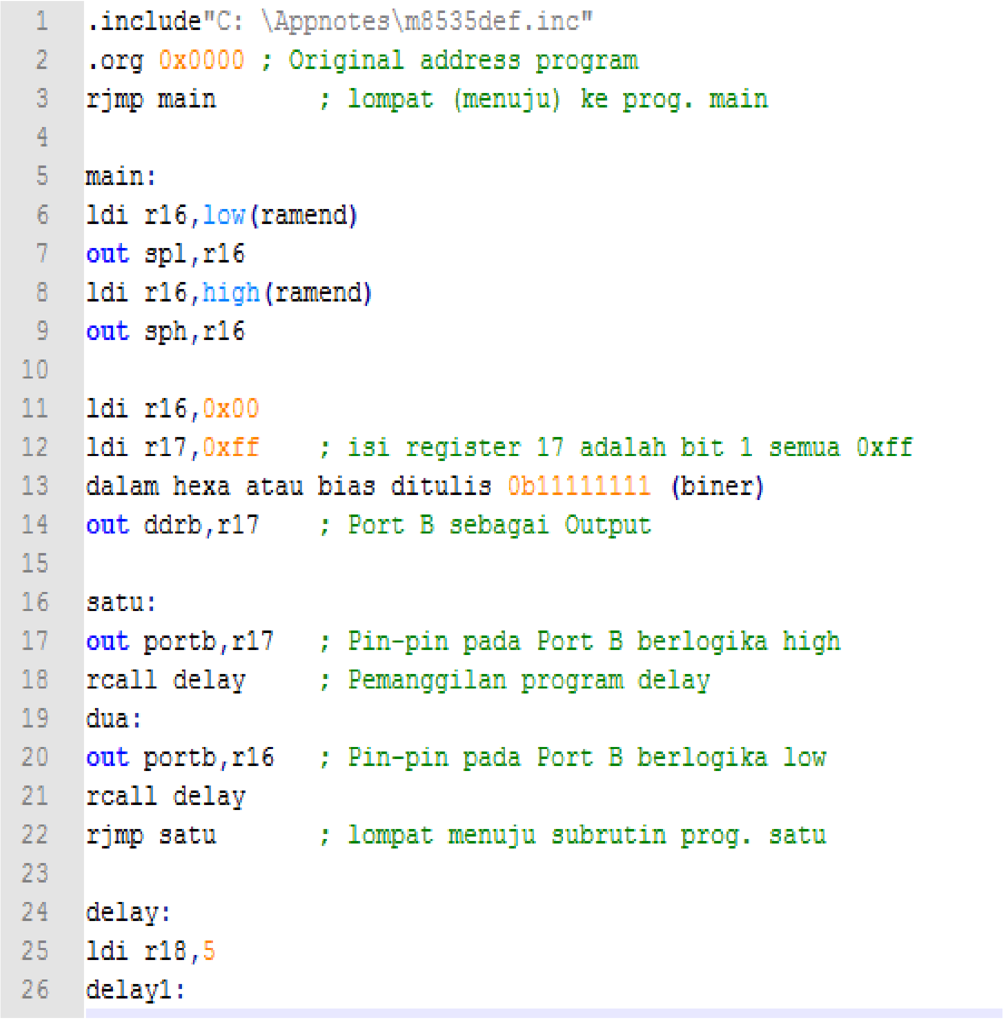
1.	SBI	P.b	Set Bit in I/O register I/O	(P,b)←1	None	2
2.	CBI	P,b	Clear Bit in I/O register I/O	(P,b)←0	None	2
3.	LSL	Rd	Logical Shift left	Rd(n+1)←Rd(n), Rd(0)←0	Z,C,N, V	1
4.	LSR	Rd	Logical Shift Right	Rd(n)←Rd(n+1), Rd(7)←0	Z,C,N, V	1
5.	ROL	Rd	Rotate Left Through Carry	Rd(0)←C, Rd(n+1)←Rd(n), C← Rd(7)	Z,C,N, V	1
6.	ROR	Rd	Rotate Right Through Carry	Rd(7)←C, Rd(n)←Rd(n+1), C← Rd(0)	Z,C,N, V	1
7.	ASR	Rd	Arithmetic Shift Right	Rd(n)←Rd(n+1), n=0…6	Z,C,N, V	1
8.	SWAP	Rd	Swap Nibbles	Rd(3..0)←Rd(7..4), Rd(7..4)←Rd(3..0)	None	1

Control Instruction



No	Instruksi	Operand	Deskripsi	Operasi	Flags	Clock
1.	NOP		No Operation		None	1
2.	SLEEP		Sleep	(see specific descr. for sleep function)	None	1
3.	WDR		Watchdog Reset	(see specific descr. for WDR/Timer)	None	1
4.	BREAK		Break	For On-chip debug only	None	N/A

Assembly Example



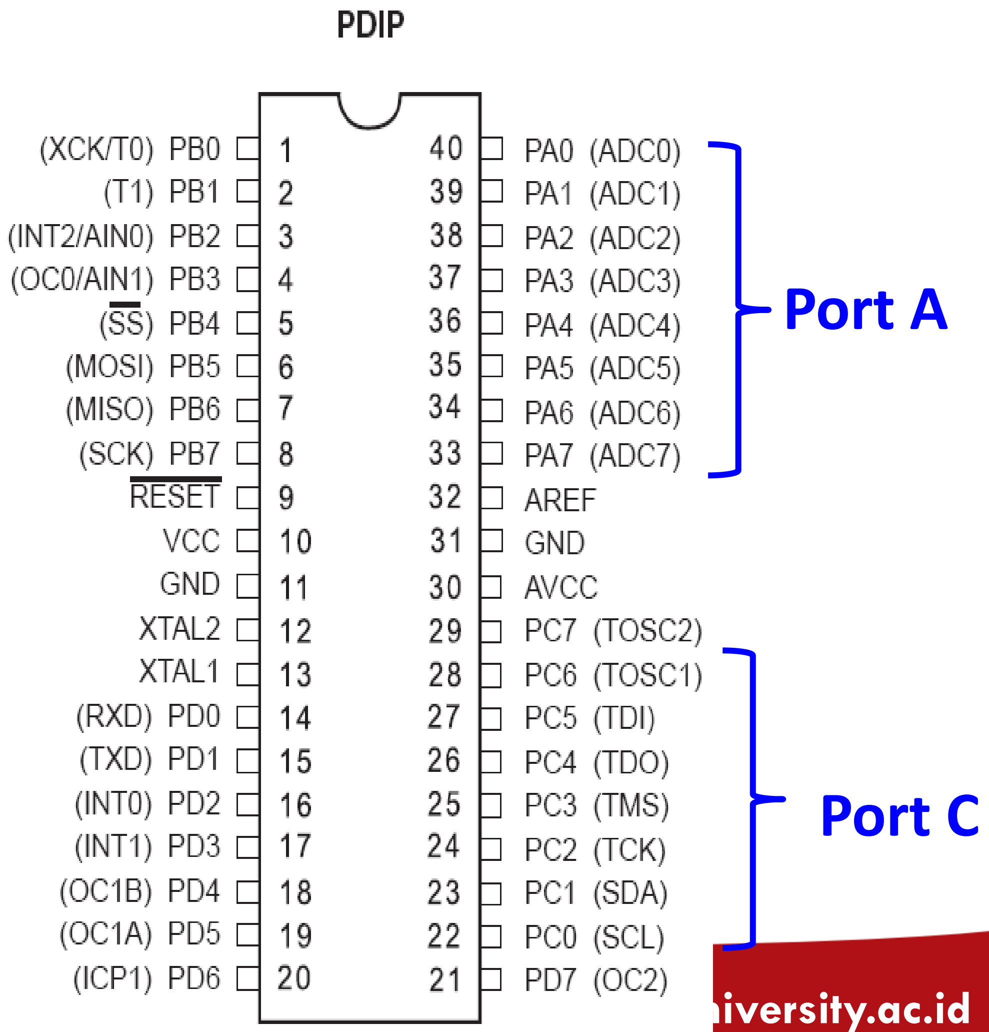
```
.include"C: \Appnotes\m8535def.inc"
.org 0x0000 ; Original address program
rjmp main       ; lompat (menuju) ke prog. main

main:
ldi r16,low(ramend)
out spl,r16
ldi r16,high(ramend)
out sph,r16

ldi r16,0x00
ldi r17,0xff    ; isi register 17 adalah bit 1 semua 0xff
dalam hexa atau bias ditulis 0b11111111 (biner)
out ddrb,r17    ; Port B sebagai Output

satu:
out portb,r17   ; Pin-pin pada Port B berlogika high
rcall delay     ; Pemanggilan program delay
dua:
out portb,r16   ; Pin-pin pada Port B berlogika low
rcall delay
rjmp satu       ; lompat menuju subrutin prog. satu

delay:
ldi r18,5
delay1:
```

Hardware Capability

ATMega 32: Pin Layout

Port B

Port D



PDIP

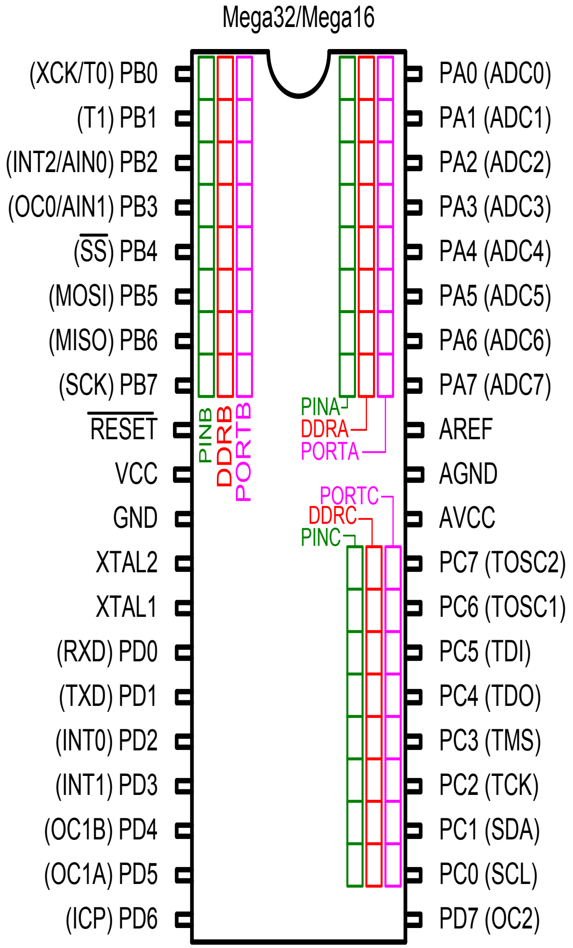
(XCK/T0) PB0	1	40	PA0 (ADC0)
(T1) PB1	2	39	PA1 (ADC1)
(INT2/AIN0) PB2	3	38	PA2 (ADC2)
(OC0/AIN1) PB3	4	37	PA3 (ADC3)
($\overline{\text{SS}}$) PB4	5	36	PA4 (ADC4)
(MOSI) PB5	6	35	PA5 (ADC5)
(MISO) PB6	7	34	PA6 (ADC6)
(SCK) PB7	8	33	PA7 (ADC7)
$\overline{\text{RESET}}$	9	32	AREF
VCC	10	31	GND
GND	11	30	AVCC
XTAL2	12	29	PC7 (TOSC2)
XTAL1	13	28	PC6 (TOSC1)
(RXD) PD0	14	27	PC5 (TDI)
(TXD) PD1	15	26	PC4 (TDO)
(INT0) PD2	16	25	PC3 (TMS)
(INT1) PD3	17	24	PC2 (TCK)
(OC1B) PD4	18	23	PC1 (SDA)
(OC1A) PD5	19	22	PC0 (SCL)
(ICP1) PD6	20	21	PD7 (OC2)

Port A

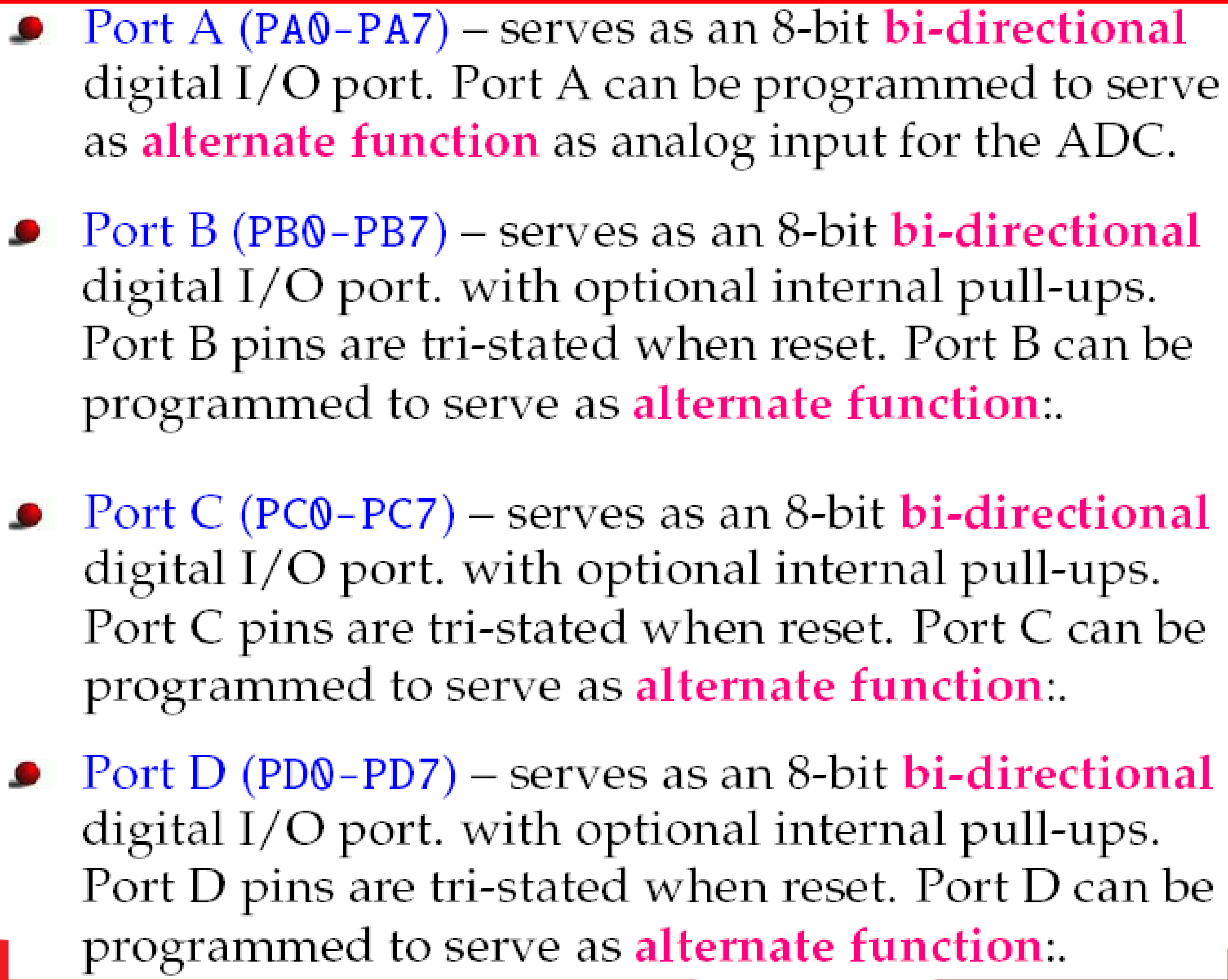
Port C

ATMega32 Pin out & Descriptions

- There are 4 ports that provide parallel I/O interfaces to outside world: **Port A, Port B, Port C & Port D.**
  - Each port provides 8 **bidirectional** digital I/O lines which are connected to ATmega32 pins provided that *alternate functions* are not selected on that port.
  - Eventhough **bidirectional**, at any time the I/O line can either be **Input or Output**.
  - The **Directions** of each I/O lines must be **configured** (input or output) before they are used.
  - Naming convention:
    - PORT*x* ≡ I/O reg for Port *x* where *x* = A, B, C, D.
    - PORT*xn* ≡ Port *x* bit *n* where *n* = 0 – 7.



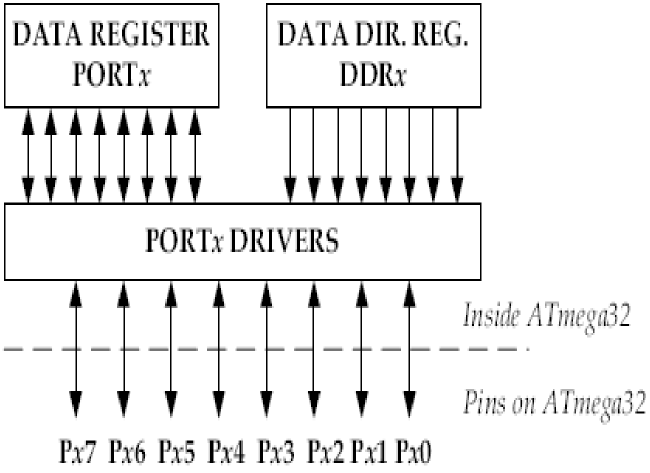
Port Description

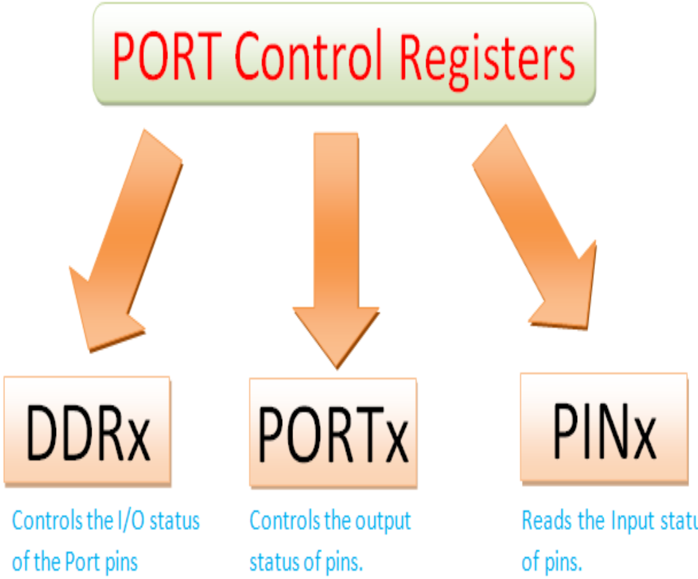


- Port A (PA0-PA7) – serves as an 8-bit **bi-directional** digital I/O port. Port A can be programmed to serve as **alternate function** as analog input for the ADC.
- Port B (PB0-PB7) – serves as an 8-bit **bi-directional** digital I/O port. with optional internal pull-ups. Port B pins are tri-stated when reset. Port B can be programmed to serve as **alternate function**:.
- Port C (PC0-PC7) – serves as an 8-bit **bi-directional** digital I/O port. with optional internal pull-ups. Port C pins are tri-stated when reset. Port C can be programmed to serve as **alternate function**:.
- Port D (PD0-PD7) – serves as an 8-bit **bi-directional** digital I/O port. with optional internal pull-ups. Port D pins are tri-stated when reset. Port D can be programmed to serve as **alternate function**:.

ATMega32 Pin out & Descriptions

- The general programmer view of Port A, B, C & D :




PORT Control Registers

DDRx

Controls the I/O status of the Port pins

PORTx

Controls the output status of pins.

PINx

Reads the Input statu of pins.

- Each Port *x* has three 8-bit **Registers** associated with it.

  Register ≈ a memory :

  - **DDR*x*** – *Data Direction Register* for Port *x* (Read/Write).
  - **PORT*x*** –*Data Register* for Port *x* (Read/Write).
  - **PIN*x*** – *Port Input Pins Register* for Port *x* (Read only).

Single pin accessing

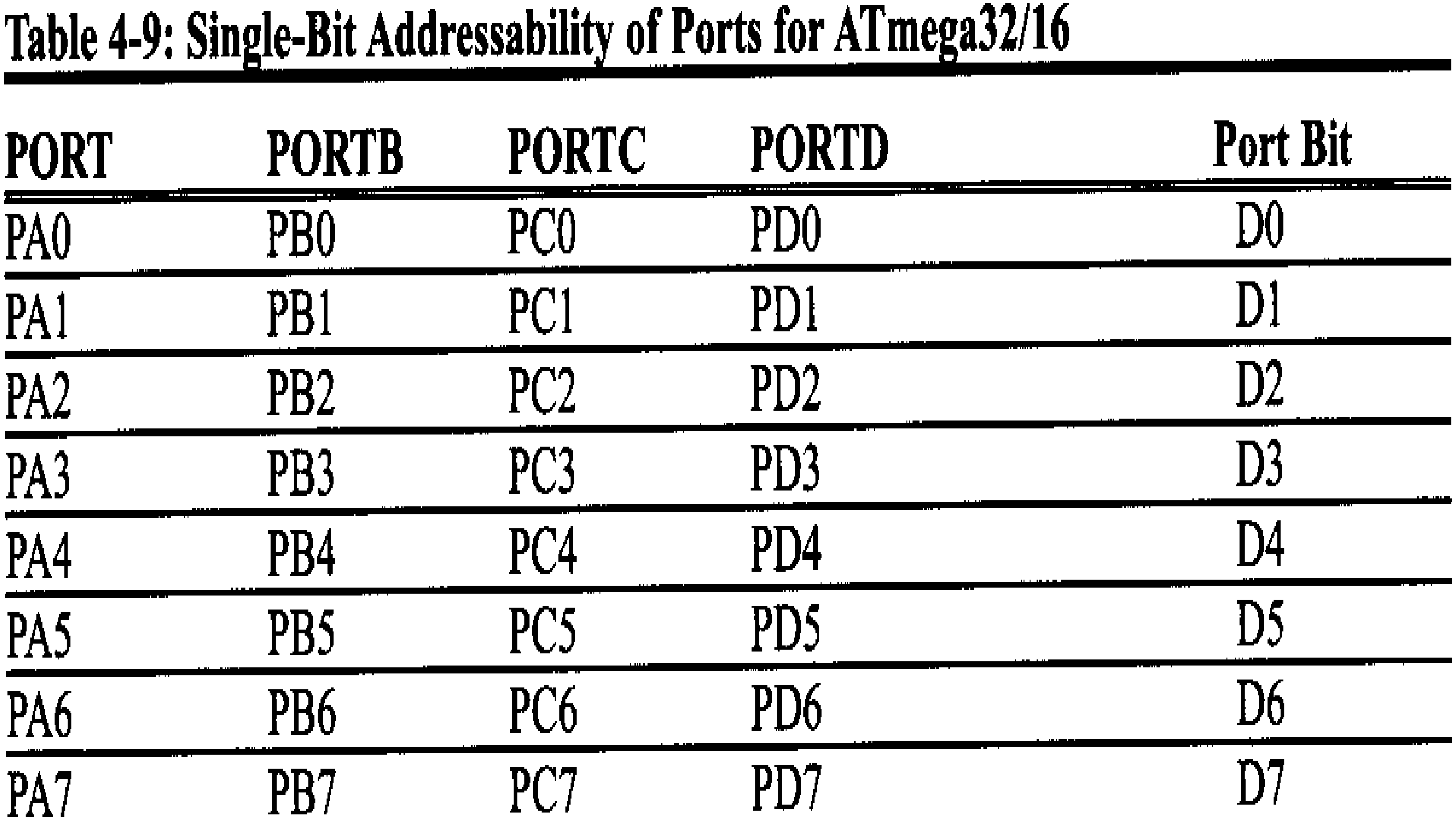
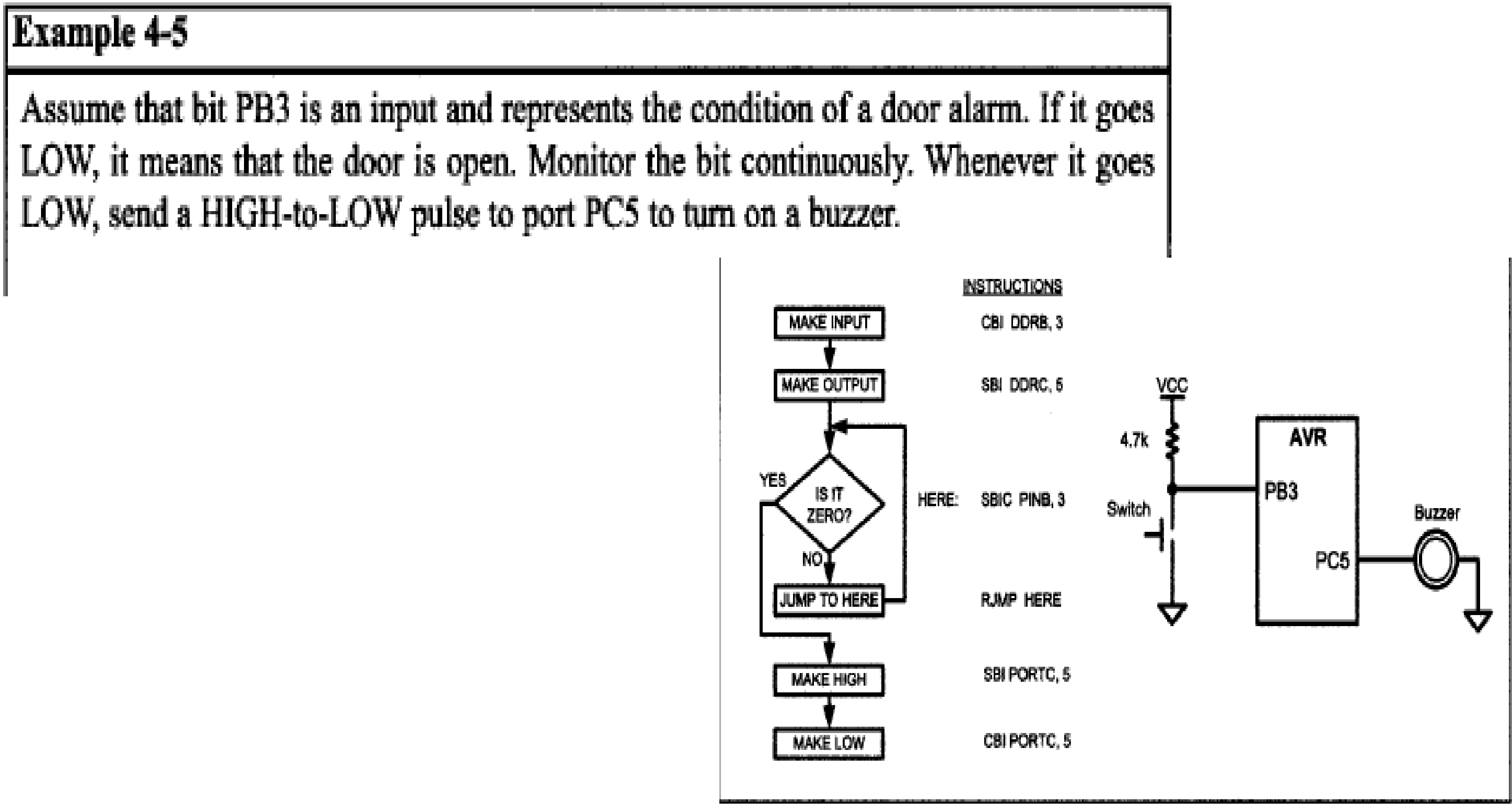


Table 4-9: Single-Bit Addressability of Ports for ATmega32/16

PORT	PORTB	PORTC	PORTD	Port Bit
PA0	PB0	PC0	PD0	D0
PA1	PB1	PC1	PD1	D1
PA2	PB2	PC2	PD2	D2
PA3	PB3	PC3	PD3	D3
PA4	PB4	PC4	PD4	D4
PA5	PB5	PC5	PD5	D5
PA6	PB6	PC6	PD6	D6
PA7	PB7	PC7	PD7	D7

Example



Example 4-5

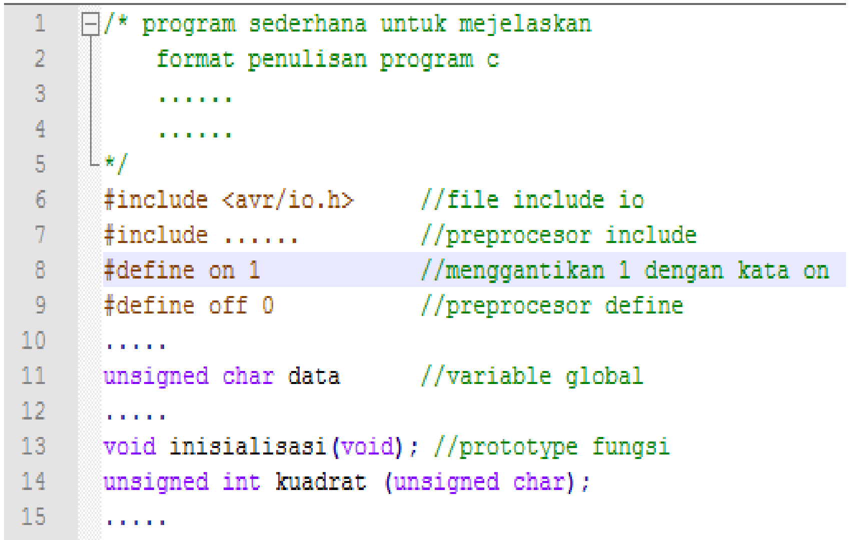
Assume that bit PB3 is an input and represents the condition of a door alarm. If it goes LOW, it means that the door is open. Monitor the bit continuously. Whenever it goes LOW, send a HIGH-to-LOW pulse to port PC5 to turn on a buzzer.

INSTRUCTIONS

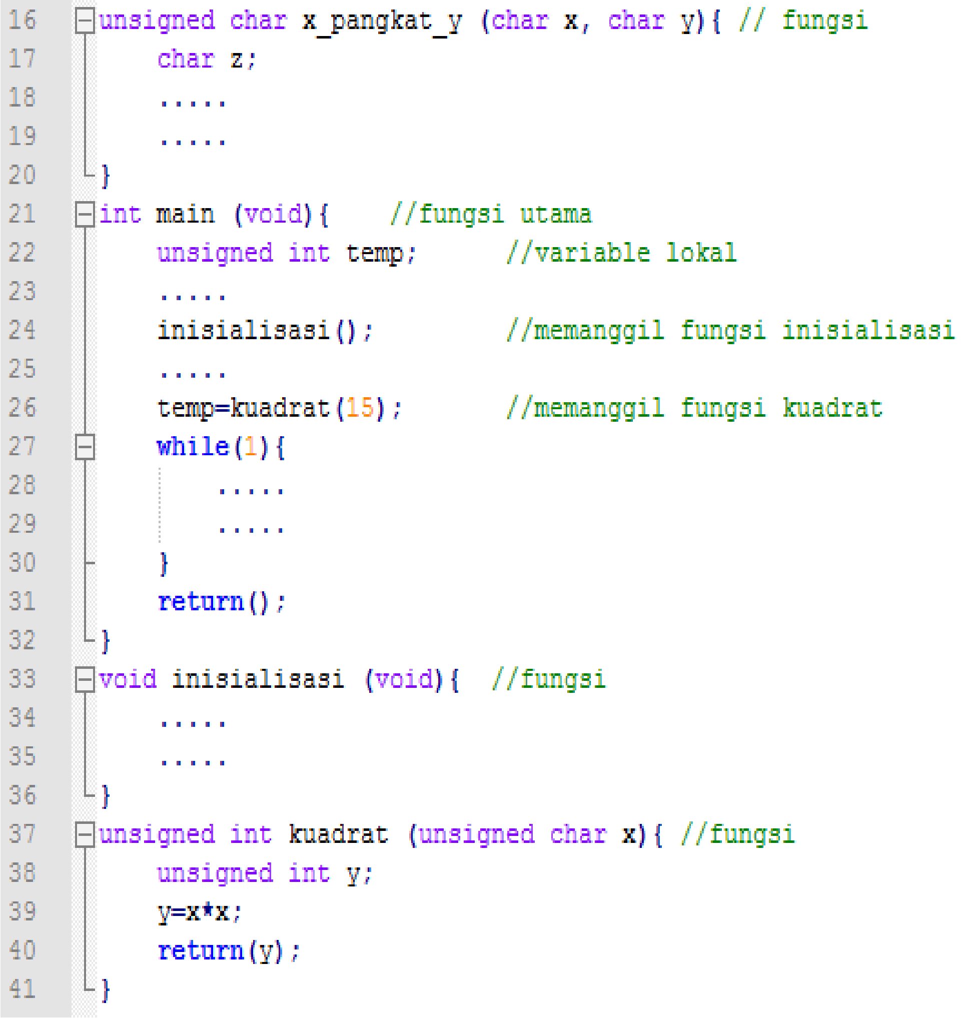
```
       CBI  DDRB, 3
       SBI  DDRC, 5
HERE:  SBIC PINB, 3
       RJMP HERE
       SBI PORTC, 5
       CBI PORTC, 5
```

Using C Language
to Program your Microcontroller

An Example of ATMEL Program in C Language



```
/* program sederhana untuk mejelaskan
    format penulisan program c
    ......
    ......
*/
#include <avr/io.h>     //file include io
#include ......         //preprocesor include
#define on 1            //menggantikan 1 dengan kata on
#define off 0           //preprocesor define
.....
unsigned char data      //variable global
.....
void inisialisasi(void); //prototype fungsi
unsigned int kuadrat (unsigned char);
.....
```



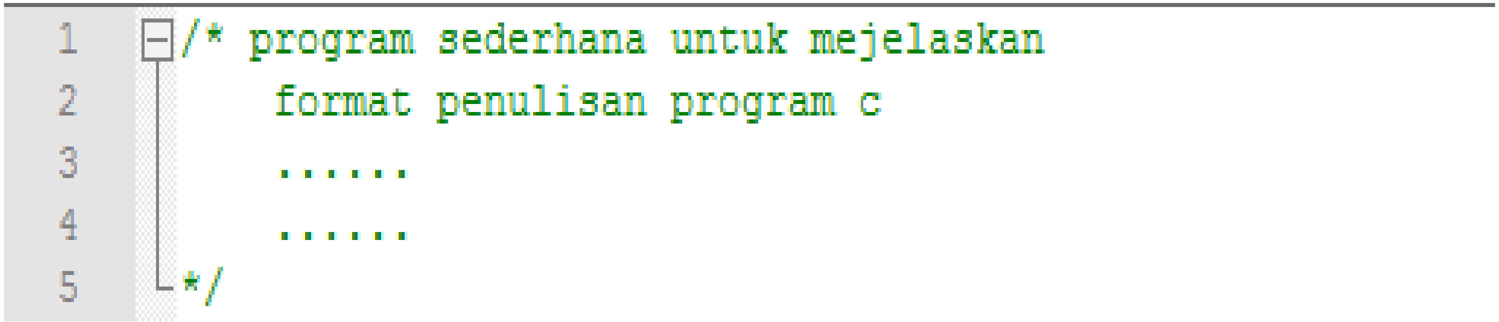
```
unsigned char x_pangkat_y (char x, char y){ // fungsi
    char z;
    .....
    .....
}
int main (void){    //fungsi utama
    unsigned int temp;      //variable lokal
    .....
    inisialisasi();         //memanggil fungsi inisialisasi
    .....
    temp=kuadrat(15);       //memanggil fungsi kuadrat
    while(1){
        .....
        .....
    }
    return();
}
void inisialisasi (void){  //fungsi
    .....
    .....
}
unsigned int kuadrat (unsigned char x){ //fungsi
    unsigned int y;
    y=x*x;
    return(y);
}
```

Comment

- Ignored by compiler but very useful for other to understand the program

  /* for writing comment in a paragraf */

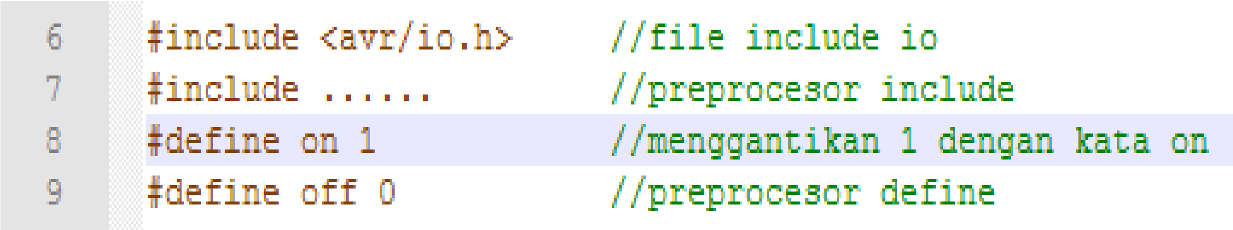
  // for writing a 1 line comment



```
/* program sederhana untuk mejelaskan
     format penulisan program c
     ......
     ......
*/
```

Preprocessor (6-10)

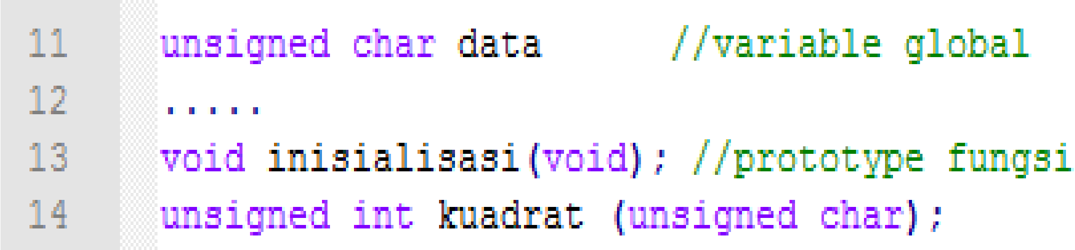
- Preprocessor #include can be used to attach a library function (h header file) so we may use many built in functions
- Header “io.h” contains definition for SFR (Special Function Register) and all pins and bits in μC
- Preprocessor #define is used for defining a constant or macro



```
#include <avr/io.h>       //file include io
#include ......           //preprocesor include
#define on 1              //menggantikan 1 dengan kata on
#define off 0             //preprocesor define
```

Variable Declaration

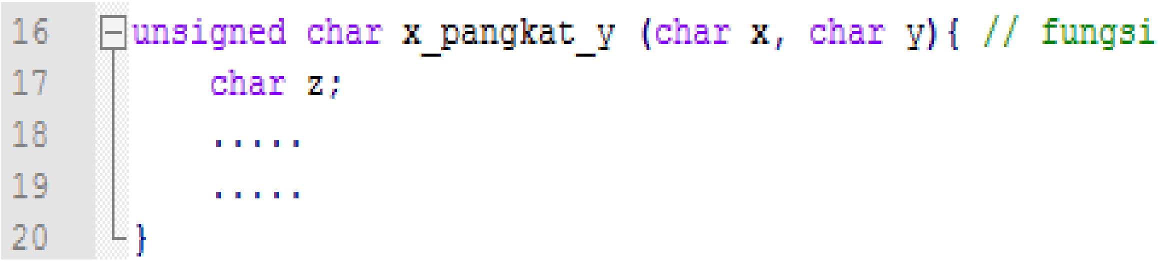
- Variable is used to store a value within a program
- A global variable is defined outside any function and can be accessed by all functions
- A local variable is defined inside a function and can only be accessed by that function
- How to define variable:
  - `DataType    VariableName;`



```
unsigned char data      //variable global
.....
void inisialisasi(void); //prototype fungsi
unsigned int kuadrat (unsigned char);
```

Function Prototype

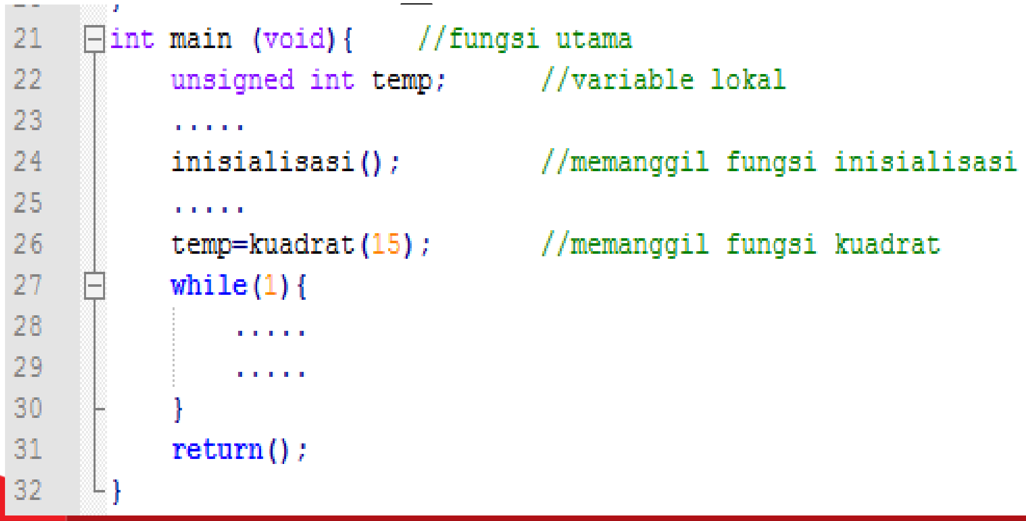
- Used to define a function to be called by other function (usually by main function)
- How to define function:
  - DataType FunctionName (DataType Parameter1, DataType Parameter 2)



```
unsigned char x_pangkat_y (char x, char y){ // fungsi
    char z;
    .....
    .....
}
```

Main Function

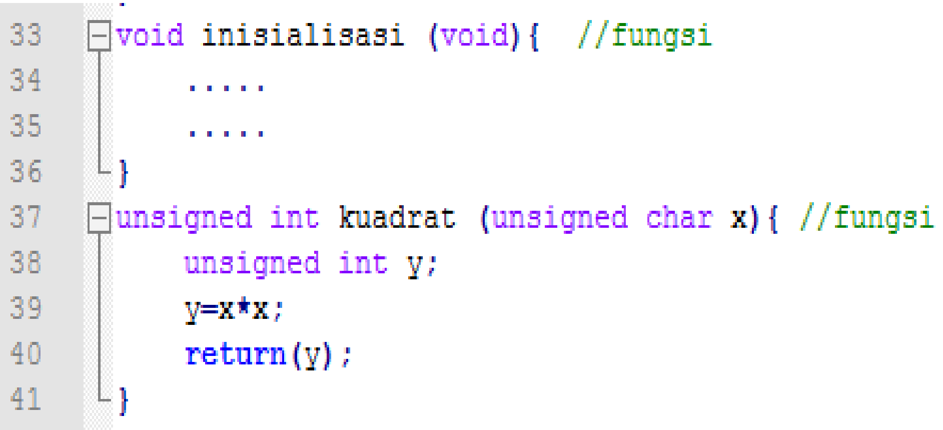
- The first function to be executed starting from the first line
- How to call a function from main function:
  - Without return value and without input parameter
    `function()`
  - With return value but without input parameter
    `variable = functionName();`
  - With return value and with input parameter
    `variable = functionName(variable_or constant)`



```
int main (void){     //fungsi utama
    unsigned int temp;      //variable lokal
    .....
    inisialisasi();         //memanggil fungsi inisialisasi
    .....
    temp=kuadrat(15);       //memanggil fungsi kuadrat
    while(1){
        .....
        .....
    }
    return();
}
```

Sub-Program or Function

- Function is a sub module to solve a specific problem (ex. calculate factorial)
- How to define a function:
  - Without return value and without input parameter
    `void function(void)`
  - With return value but without input parameter
    `DataType functionName(void);`
  - With return value and with input parameter
    `DataType functionName(DataType parameter1, ...)`



```
void inisialisasi (void){  //fungsi
    .....
    .....
}
unsigned int kuadrat (unsigned char x){ //fungsi
    unsigned int y;
    y=x*x;
    return(y);
}
```

Data Type

Data Type	Byte	Bit	min	Max
char	1	8	-128	127
signed char	1	8	-128	127
unsigned char	1	8	0	255
Int	2	16	-32768	32767
signed int	2	16	-32768	32767
unsigned int	2	16	0	65535
long	4	32	-2147483648	2147483647
signed long	4	32	-2147483648	2147483647
unsigned long	4	32	0	4294967295
float	4	32	1,28E-38	3,4E38

ATMEL Studio

- Please try to explore the ATMEL Studio using provided example projects

See you on next class