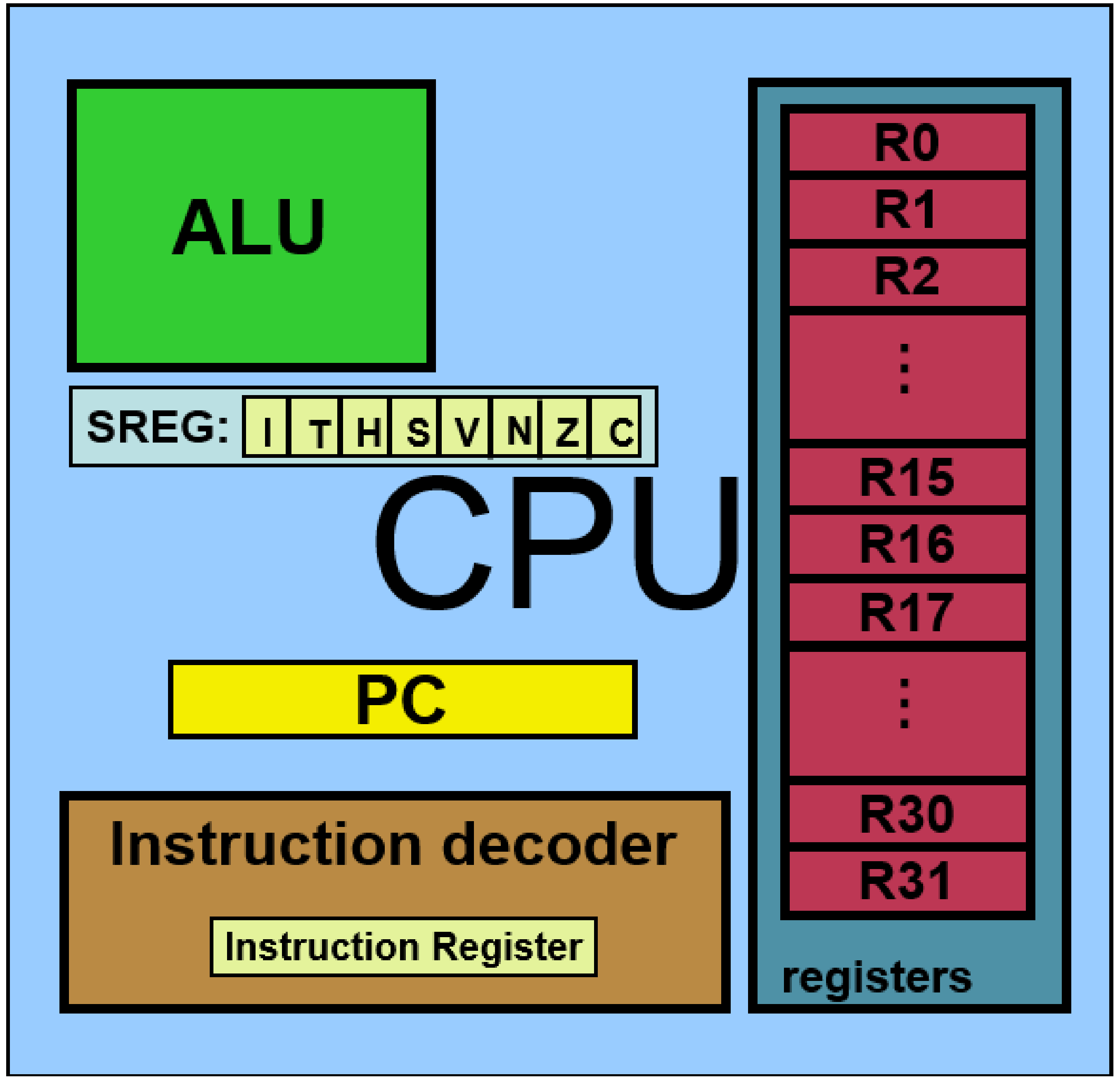
Outline

- ❑ Arduino AVR Architecture
- ❑ Assembly Language Programming
- ❑ Examples of Assembly Programming
- ❑ Flow Chart Drawing and Algorithm Development

AVR Architecture of CPU



- Arithmetic Logic Unit (ALU)
- 32 General Purpose Register (GPR) ← each 8-bit
- Program Counter (PC)
- Instruction Decoder
- Status Register (SREG)

AVR Flags/Status Register

- C = Carry Flag, This flag is set whenever there is a carry out from the D7 bit after an arithmetic operation(Addition, subtraction, increment, decrement, etc.). This flag bit is affected after an 8-bit addition or substruction.
- Z = Zero Flag, This flag is affected after an arithmetic or logic operation. If the result is zero than Z = 1, else Z = 0.
- N = Negative Flag, It reflects the result of an arithmetic operation. If the D7 bit of the result is zero, then N = 0 and the result is positive. Else N = 1 and the result is negative.
- V = Overflow Flag,
- S = Sign Flag,
- H = Half Carry Flag, this bit is set if there is a carry from D3 to D4 bit after ADD or SUB instruction.
- T = Bit Copy Storage. Used as a temporary storage for bit. It can be used to copy a bit from a GPR to another GPR.
- I = Global Interrupt Enable

Direct Data Register Method

- To assign input- **0**
- To assign output- **1**
- DDRB= 0x 05;
- Or DDRB= 0b 00000101; PB0 and PB2 are output pin while rest are input.
- To write High-1
- To write Low-0
- PORTB= 0x01;
- PORTB= 0b00000001; write PB0 as high.

PORTB Register

PB7	PB6	PB5	PB4	PB3	PB2	PB1	PB0
0	0	0	0	0	0	0	1

DDRB Register

PB7	PB6	PB5	PB4	PB3	PB2	PB1	PB0
0	0	0	0	0	1	**0**	**1**

Instruction Set Summary

Mnemonics	Operands	Description	Operations	Flags	Clocks
ADD	Rd,Rr	Add two Registers	Rd ← Rd + Rr	Z,C,N,V,H	1
ADC	Rd,Rr	Add with Carry two Registers	Rd ← Rd + Rr + C	Z,C,N,V,H	1
SUB	Rd,Rr	Subtract two Registers	Rd ← Rd - Rr	Z,C,N,V,H	1
SUBI	Rd,K	Subtract Constant from Register	Rd ← Rd - K	Z,C,N,V,H	1
AND	Rd,Rr	Logical AND Registers	Rd ← Rd • Rr	Z,N,V	1
OR	Rd,Rr	Logical OR Registers	Rd ← Rd V Rr	Z,N,V	1
NEG	Rd	Two’s Complement	Rd ← 0x00 - Rd	Z,C,N,V,H	1
INC	Rd	Increment	Rd ← Rd + 1	Z,N,V	1
DEC	Rd	Decrement	Rd ← Rd - 1	Z,N,V	1
MUL	Rd,Rr	Multiply Unsigned	R1:R0 ← Rd x Rr	Z,C	2

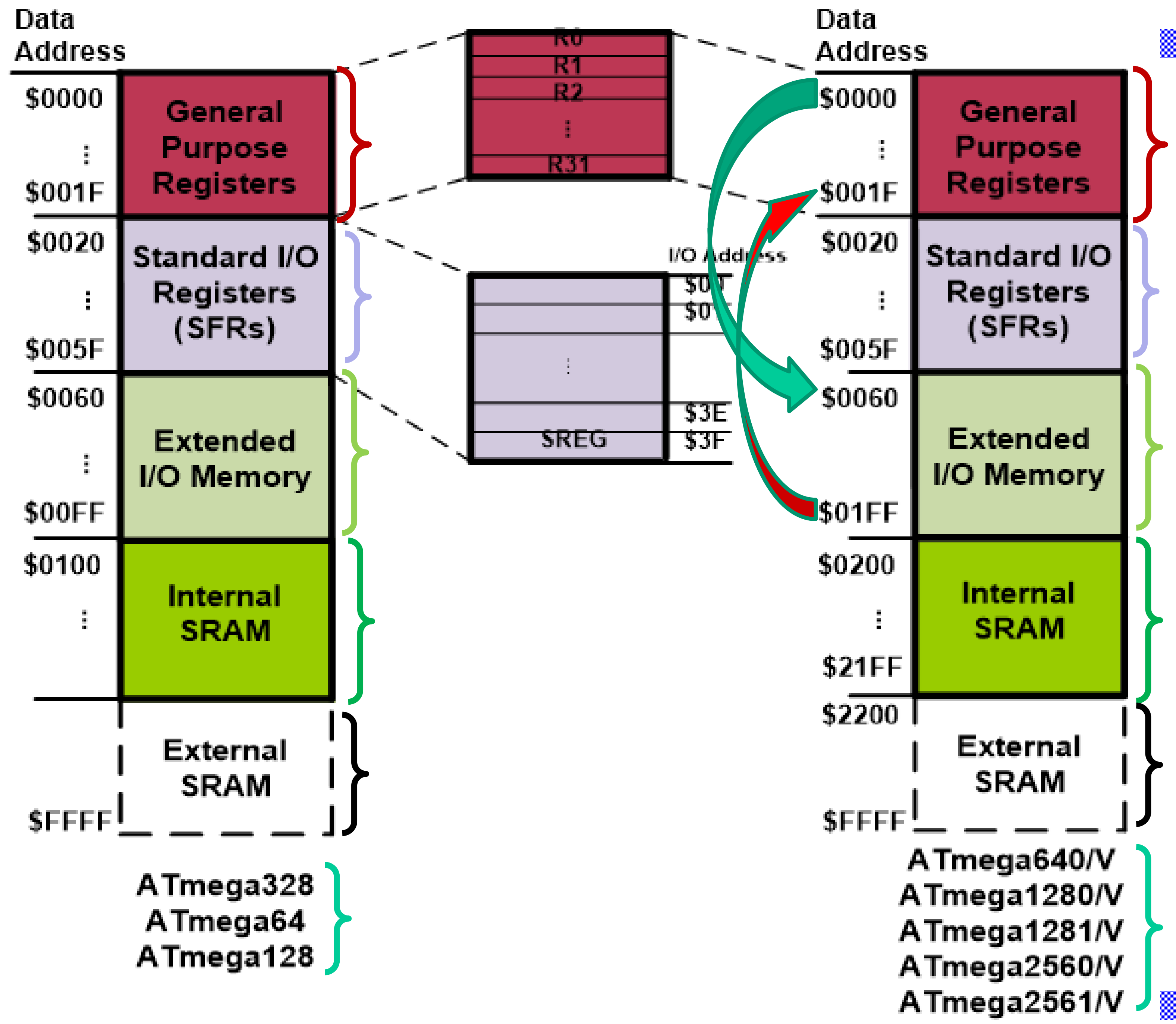
Instruction Set Summary

Mnemonics	Operands	Description	Operations	Flags	Clocks
TST	Rd	Test for Zero or Minus	Rd ← Rd • Rd	Z,N,V	1
CLR	Rd	Clear Register	Rd ← Rd ⊕ Rd	Z,N,V	1
MOV	Rd, Rr	Move Between Registers	Rd ← Rr	None	1
LDI	Rd, K	Load Immediate	Rd ← K	None	1
LDS	Rd, K	Load Direct from data space location K	Rd← [K]	None	1
IN	Rd, P	From In Port P/ address to Rd register	Rd ← P	None	1
OUT	P, Rr	From Rr register to Out Port P/ address	P ← Rr	None	1
PUSH	Rr	Push Register on Stack	STACK ← Rr	None	2
POP	Rd	Pop Register from Stack	Rd ← STACK	None	2
NOP		No Operation	None	1	
BREAK		Break	For On-chip Debug Only	None	N/A

Instruction Set Summary

Mnemonics	Operands	Description	Operations	Flags	Clocks
STS	K, Rr	Store Rr to data space location K.	[K] ← Rr	None	1
SBI	A, b	Sets a specified bit in an I/O Register	Sets the b no. bits of the A register	None	1
CBI	A, b	Resets a specified bit in an I/O Register	Resets the b no bits of the A register	None	1
SBIC	A, b	Skip if Bit in I/O Register is Cleared/ Reset	If b no bit of A register is 0, then skip PC by 2 or 3.	None	1
SBIS	A, b	Skip if Bit in I/O Register is set	If b no bit of A register is 1, then skip PC by 2 or 3.	None	1
RJUMP	K	Relative jump	PC← PC+K+1	None	1

Memory Locations



STS 0x60, R0

LDS R31, 0x01FF

Register Contents

SBI DDRB, 3;

DDRB Register

PB7	PB6	PB5	PB4	PB3	PB2	PB1	PB0
0	0	0	0	0	1	0	0

CBI DDRB, 5;

DDRB Register

PB7	PB6	PB5	PB4	PB3	PB2	PB1	PB0
0	0	0	0	0	0	0	0

SBIC R7, 5;

R7 Register

8	7	6	5	4	3	2	1
1	1	1	0	1	1	1	1

PC = PC+2 or 3

SBIS R8, 5;

R8 Register

8	7	6	5	4	3	2	1
1	1	1	1	1	1	1	1

PC = PC+2 or 3

Programming Examples

Example: State the contents of R20, R21, and data memory location of 0x120 after executing the following program.

```
LDI R20, 5;
LDI R21, 2;
ADD R20,R21;
ADD R20,R21;
STS 0x120, R20;
```

Solution:

```
LDI R20, 5;          R20=5
LDI R21, 2;          R21=2
ADD R20, R21;        R20=5+2=7
ADD R20, R21;        R20= 7+2=9
STS 0x120, R20;      0x120 = 9
```

R20= 9
R21= 2
0x120= 9

Example: State the contents of RAM locations of $212, $213, $214, $215, and $216 after executing the following program.

```
LDI R16, 0x99
STS 0x212, R16
LDI R16, 0x85
STS 0x213, R16
LDI R16, 0x3F
STS 0x214, R16
LDI R16, 0x63
STS 0x215, R16
LDI R16, 0x12
STS 0x216, R16
```

Solution:	
0x212	0x99
0x213	0x85
0x214	0x3F
0x215	0x63
0x216	0x12

WHERE LEADERS ARE CREATED

Difference Between Assembly and C

Assembly Code Example
sei *; set Global Interrupt Enable* **sleep***; enter sleep, waiting for interrupt* ; note: will enter sleep before any pending interrupt(s)
C Code Example
__enable_interrupt(); */* set Global Interrupt Enable */* __sleep(); */* enter sleep, waiting for interrupt */* /* note: will enter sleep before any pending interrupt(s) */

EEPROM Control Register (EECR)

7	6	5	4	3	2	1	0
		EEPM(1:0)		EERIE	EEMPE	EEPE	EERE

SBIC EECR, EEPE; If EEPE bit of EECR is 0 then the Program Counter (PC) will skip 2 or 3 steps.

Difference Between Assembly and C

Assembly Code Example	C Code Example

EEPROM_write:
; Wait for completion of previous write
sbic EECR,EEPE
rjmp EEPROM_write
; Set up address (r18:r17) in address register
out EEARH, r18
out EEARL, r17
; Write data (r16) to Data Register
out EEDR,r16
; Write logical one to EEMPE
sbi EECR,EEMPE
; Start eeprom write by setting EEPE
sbi EECR,EEPE
ret
	void EEPROM_write(**unsigned int** uiAddress, **unsigned char** ucData)
	{
	/ Wait for completion of previous write */*
	while(EECR & (1<<EEPE))
	;
	/* Set up address and Data Registers */
	EEAR = uiAddress;
	EEDR = ucData;
	/ Write logical one to EEMPE */*
	EECR |= (1<<EEMPE);
	/ Start eeprom write by setting EEPE */*
	EECR |= (1<<EEPE);
	}

WHERE LEADERS ARE CREATED

Functions of Assembly and C

The next code examples show assembly and C functions for reading the EEPROM.

The examples assume that interrupts are controlled so that no interrupts will occur during the execution of these functions.

Assembly Code Example

```
EEPROM_read:
; Wait for completion of previous write
sbic EECR,EEPE
rjmp EEPROM_read
; Set up address (r18:r17) in address
register
out EEARH, r18
out EEARL, r17
; Start eeprom read by writing EERE
sbi EECR,EERE
; Read data from Data Register
in r16,EEDR
ret
```

C Code Example

```
unsigned char
EEPROM_read(unsigned int uiAddress)
{
/* Wait for completion of previous write */
while(EECR & (1<<EEPE))
;
/* Set up address register */
EEAR = uiAddress;
/* Start eeprom read by writing EERE */
EECR |= (1<<EERE);
/* Return data from Data Register */
return EEDR;
}
```

Functions of Assembly and C

Bit 0 – IVCE: Interrupt Vector Change Enable

The IVCE bit must be written to logic one to enable change of the IVSEL bit. IVCE is cleared by hardware four cycles after it is written or when IVSEL is written. Setting the IVCE bit will disable interrupts,

Assembly Code Example	C Code Example

`Move_interrupts:`
 `; Enable change of Interrupt Vectors`
 `ldi r16, (1<<IVCE)`
 `out MCUCR, r16`
 `; Move interrupts to Boot Flash section`
 `ldi r16, (1<<IVSEL)`
 `out MCUCR, r16`
 `ret`
	`void Move_interrupts(void)`
	 `{`
	 `/* Enable change of Interrupt Vectors */`
	 `MCUCR = (1<<IVCE);`
	 `/* Move interrupts to Boot Flash section */`
	 `MCUCR = (1<<IVSEL);`
	 `}`

WHERE LEADERS ARE CREATED

Functions of Assembly and C

The assembly code example returns the TCNT1 value in the r17:r16 register pair.

If an interrupt occurs between the two instructions accessing the 16-bit register, the interrupt code updates the temporary register by accessing the same or any other of the 16-bit Timer Registers, then the result of the access outside the interrupt will be corrupted. Therefore, when both the main code and the interrupt code update the temporary register, the main code must disable the interrupts during the 16-bit access.

Assembly Code Example

```
; Set TCNT1 to 0x01FF
ldi r17,0x01
ldi r16,0xFF
out TCNT1H,r17
out TCNT1L,r16
; Read TCNT1 into r17:r16
in r16,TCNT1L
in r17,TCNT1H
...
```

C Code Example

```
unsigned int i;
...
/* Set TCNT1 to 0x01FF */
TCNT1 = 0x1FF;
/* Read TCNT1 into i */
i = TCNT1;
...
```

WHERE LEADERS ARE CREATED

Functions of Assembly and C

The following code examples show how to do an atomic read of the TCNT1 Register contents.

Reading any of the OCR1A/B or ICR1 Registers can be done by using the same principle

Assembly Code Example

```
TIM16_ReadTCNT1:
; Save global interrupt flag
in r18,SREG
; Disable interrupts
cli
; Read TCNT1 into r17:r16
in r16,TCNT1L
in r17,TCNT1H
; Restore global interrupt flag
out SREG,r18
ret
```

C Code Example

```
uunsigned int TIM16_ReadTCNT1( void )
{
unsigned char sreg;
unsigned int i;
/* Save global interrupt flag */
sreg = SREG;
/* Disable interrupts */
_CLI();
/* Read TCNT1 into i */
i = TCNT1;
/* Restore global interrupt flag */
SREG = sreg;
return i;
```

Functions of Assembly and C

The assembly code example returns the TCNT1 value in the r17:r16 register pair.

The following code examples show how to do an atomic write of the TCNT1 Register contents. Writing any of the OCR1A/B or ICR1 Registers can be done by using the same principle

Assembly Code Example

```
TIM16_ReadTCNT1:
; Save global interrupt flag
in r18,SREG
; Disable interrupts
cli
; Read TCNT1 into r17:r16
in r16,TCNT1L
in r17,TCNT1H
; Restore global interrupt flag
out SREG,r18
ret
```

C Code Example

```
unsigned int TIM16_ReadTCNT1( void )
{
unsigned char sreg;
unsigned int i;
/* Save global interrupt flag */
sreg = SREG;
/* Disable interrupts */
_CLI();
/* Read TCNT1 into i */
i = TCNT1;
/* Restore global interrupt flag */
SREG = sreg;
return i;
}
```

Functions of Assembly and C

The assembly code example requires that the r17:r16 register pair contains the value to be written to TCNT1.

Assembly Code Example

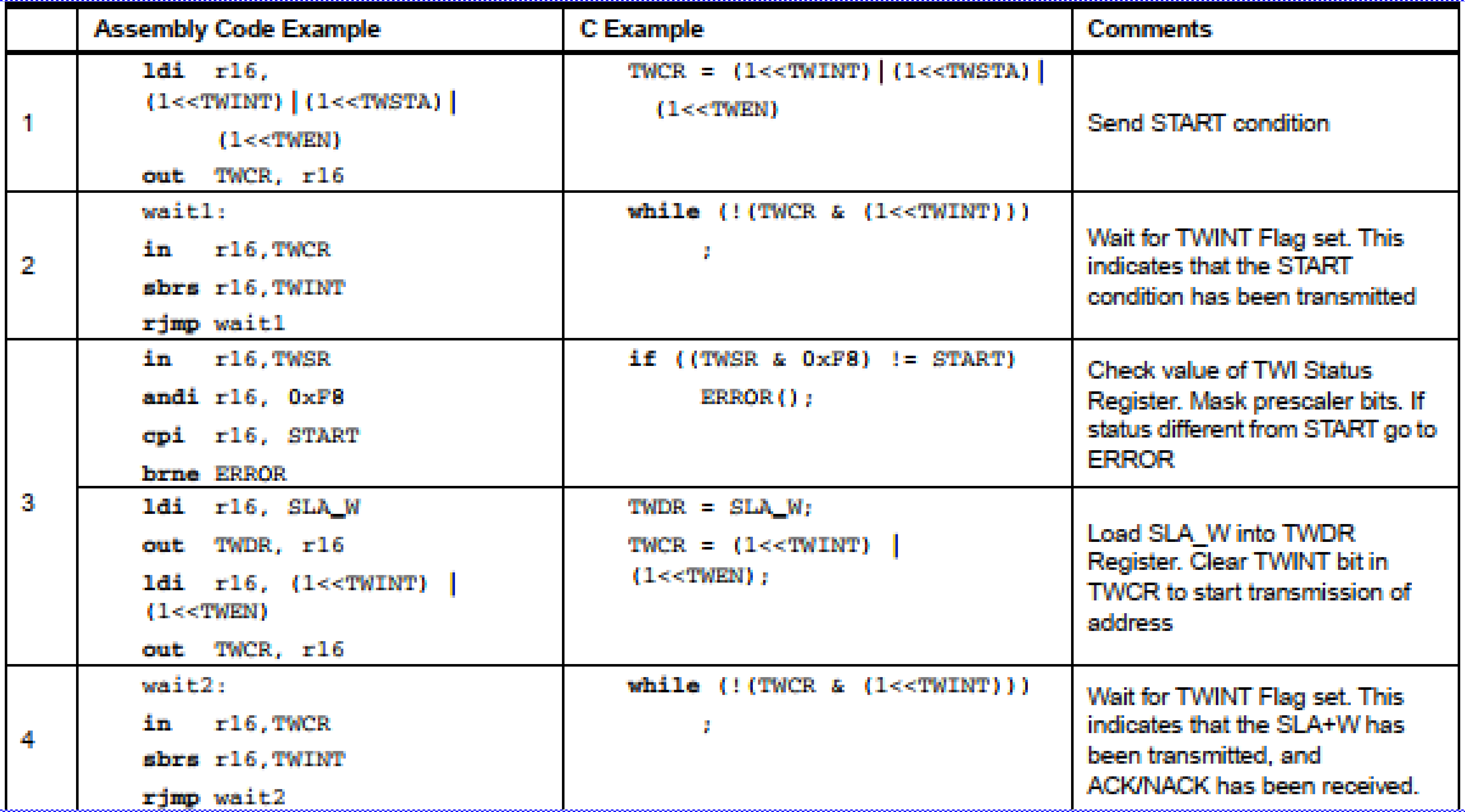
```
TIM16_WriteTCNT1:
; Save global interrupt flag
in r18,SREG
; Disable interrupts
cli
; Set TCNT1 to r17:r16
out TCNT1H,r17
out TCNT1L,r16
; Restore global interrupt flag
out SREG,r18
ret
```

C Code Example

```
void TIM16_WriteTCNT1( unsigned int i )
{
unsigned char sreg;
unsigned int i;
/* Save global interrupt flag */
sreg = SREG;
/* Disable interrupts */
_CLI();
/* Set TCNT1 to i */
TCNT1 = i;
/* Restore global interrupt flag */
SREG = sreg;
}
```

WHERE LEADERS ARE CREATED

Difference Between Assembly and C



	Assembly Code Example	C Example	Comments

1	`ldi r16, (1<<TWINT)|(1<<TWSTA)| (1<<TWEN)`
	`out TWCR, r16`
		`TWCR = (1<<TWINT)|(1<<TWSTA)| (1<<TWEN)`	Send START condition

2	`wait1:`
	`in r16,TWCR`
	`sbrs r16,TWINT`
	`rjmp wait1`
		`while (!(TWCR & (1<<TWINT))) ;`	Wait for TWINT Flag set. This indicates that the START condition has been transmitted

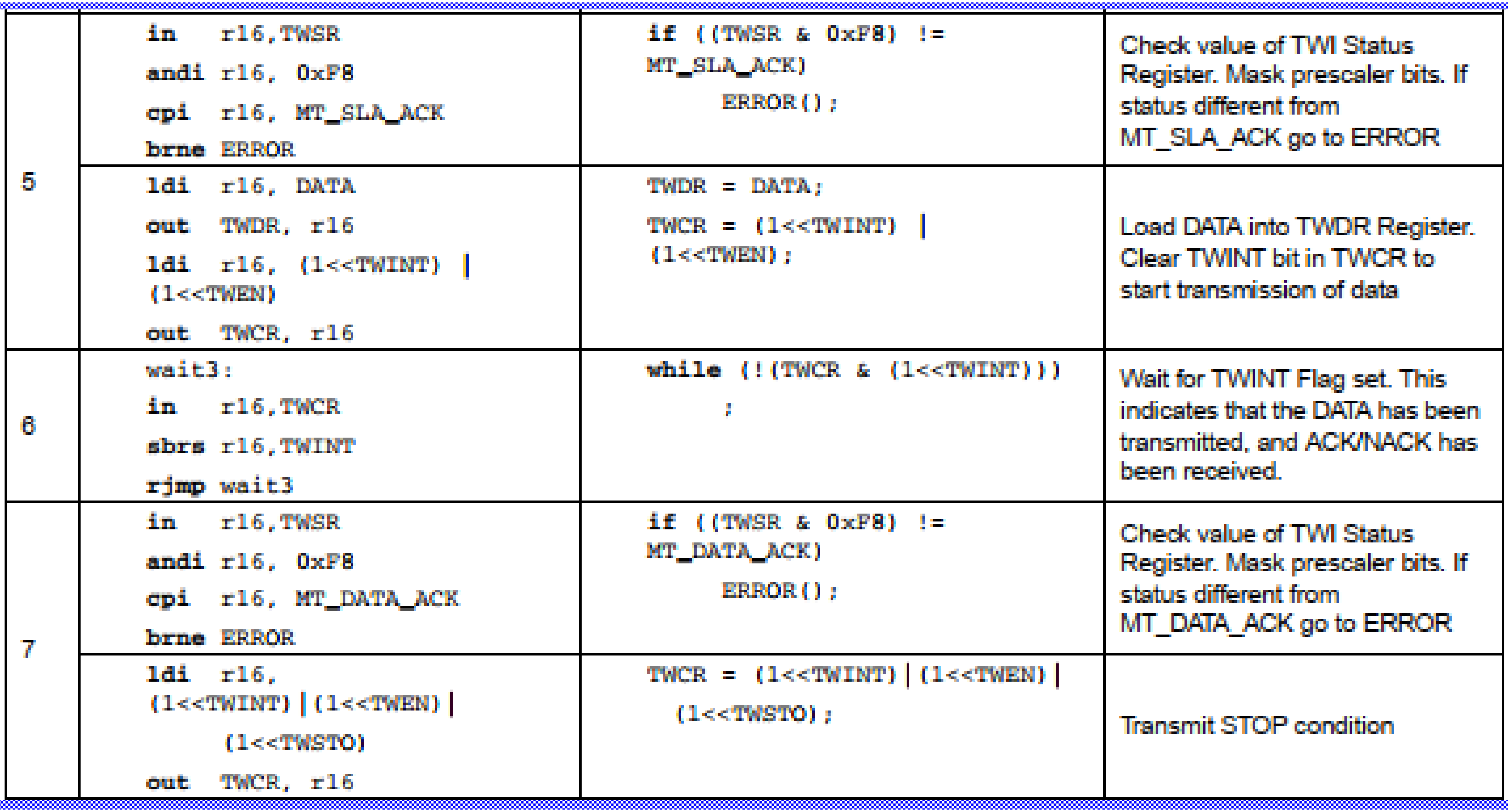
3	`in r16,TWSR`
	`andi r16, 0xF8`
	`cpi r16, START`
	`brne ERROR`
		`if ((TWSR & 0xF8) != START)`
		`ERROR();`
			Check value of TWI Status Register. Mask prescaler bits. If status different from START go to ERROR

	`ldi r16, SLA_W`
	`out TWDR, r16`
	`ldi r16, (1<<TWINT) | (1<<TWEN)`
	`out TWCR, r16`
		`TWDR = SLA_W;`
		`TWCR = (1<<TWINT) | (1<<TWEN);`
			Load SLA_W into TWDR Register. Clear TWINT bit in TWCR to start transmission of address

4	`wait2:`
	`in r16,TWCR`
	`sbrs r16,TWINT`
	`rjmp wait2`
		`while (!(TWCR & (1<<TWINT))) ;`	Wait for TWINT Flag set. This indicates that the SLA+W has been transmitted, and ACK/NACK has been received.

WHERE LEADERS ARE CREATED

Difference Between Assembly and C



5	`in r16,TWSR`
	`andi r16, 0xF8`
	`cpi r16, MT_SLA_ACK`
	`brne ERROR`
		`if ((TWSR & 0xF8) != MT_SLA_ACK)`
		`ERROR();`
			Check value of TWI Status Register. Mask prescaler bits. If status different from MT_SLA_ACK go to ERROR

	`ldi r16, DATA`
	`out TWDR, r16`
	`ldi r16, (1<<TWINT) | (1<<TWEN)`
	`out TWCR, r16`
		`TWDR = DATA;`
		`TWCR = (1<<TWINT) | (1<<TWEN);`
			Load DATA into TWDR Register. Clear TWINT bit in TWCR to start transmission of data

6	`wait3:`
	`in r16,TWCR`
	`sbrs r16,TWINT`
	`rjmp wait3`
		`while (!(TWCR & (1<<TWINT)))`
		`;`
			Wait for TWINT Flag set. This indicates that the DATA has been transmitted, and ACK/NACK has been received.

7	`in r16,TWSR`
	`andi r16, 0xF8`
	`cpi r16, MT_DATA_ACK`
	`brne ERROR`
		`if ((TWSR & 0xF8) != MT_DATA_ACK)`
		`ERROR();`
			Check value of TWI Status Register. Mask prescaler bits. If status different from MT_DATA_ACK go to ERROR

	`ldi r16, (1<<TWINT)|(1<<TWEN)| (1<<TWSTO)`
	`out TWCR, r16`
		`TWCR = (1<<TWINT)|(1<<TWEN)| (1<<TWSTO);`	Transmit STOP condition

Thanks for attending….