

SCI
Synchronize Communication Interface
RS-232
in S12

Asynchronous Serial Communication SCI

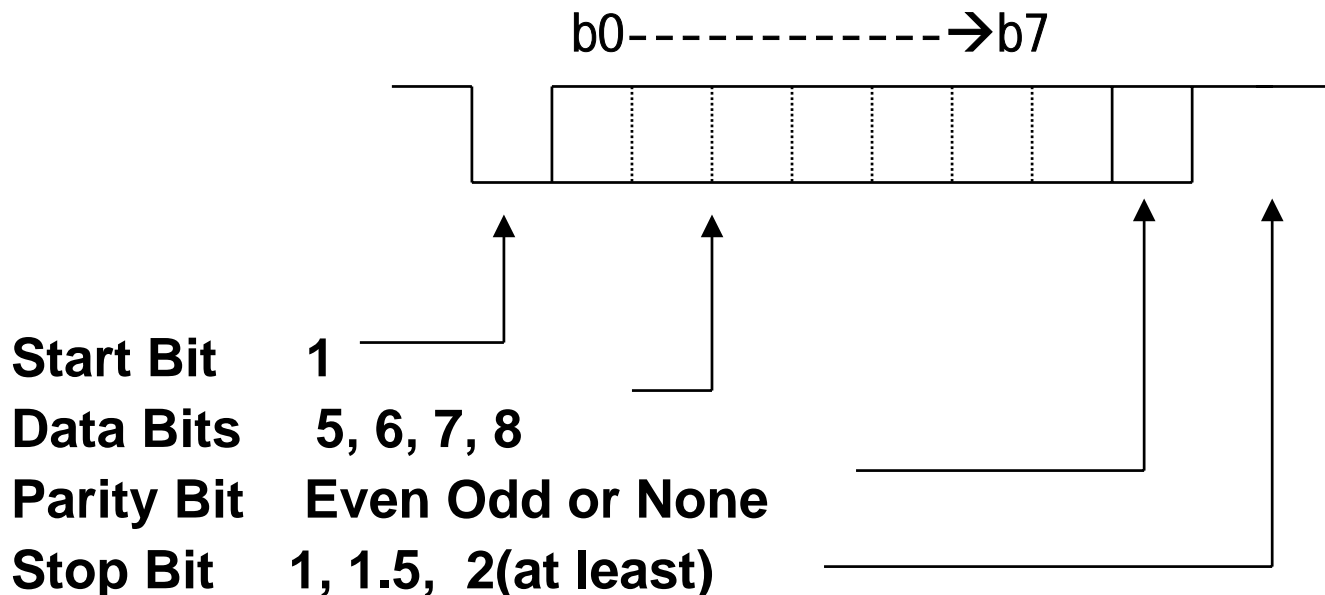
BUAD RATE:

1 Bit / Sec = 1 Buad (Bit Per Second, BPS)

standard rate(very wide bound wades):

110, 300, 600, 1200, 2400, 4800, 9600, 19200, 38400...115200

data format:



Asynchronies Serial Communication

Use Not Return to Zero (NRZ) code

T & R is synchronized by start bit detection

R Clock = T Clock x 16 (baud rate x 16), slow

3,5,7 clock detect fowling edge of start bit

8,9,10 detect bit's value (Anti-Noise)

The maximum speed: bus rate /16

Duplex and Half Duplex

Asynchronous Serial Communication and RS - 232C

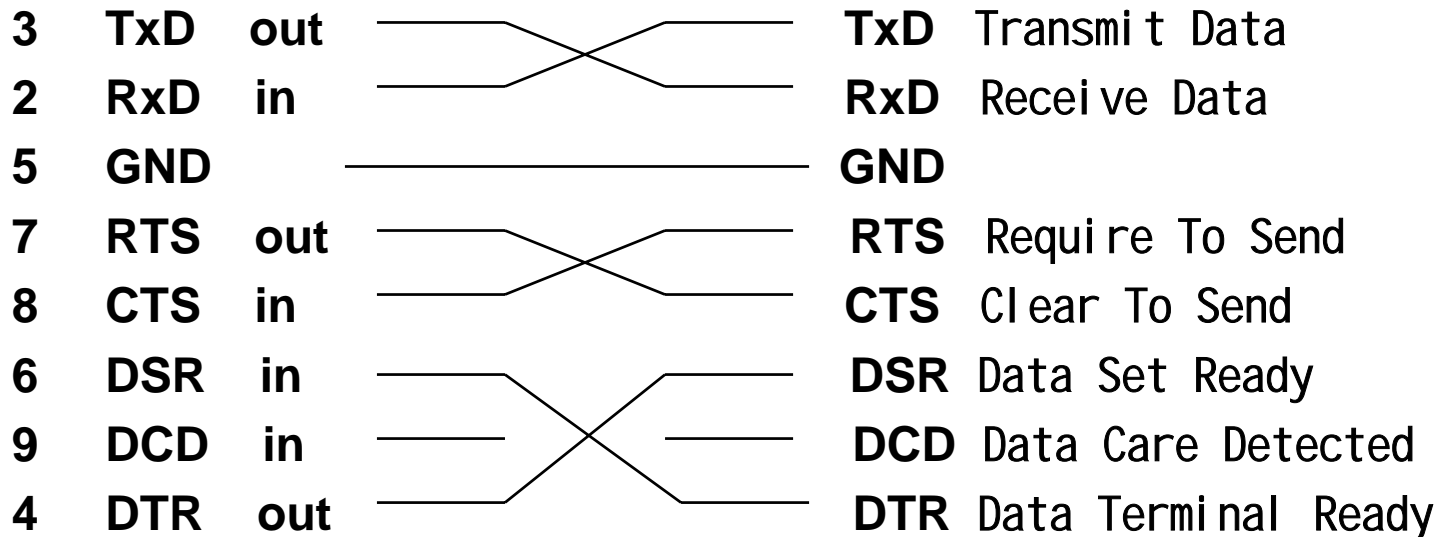
- Signals Voltage Levels

1 = -3V ~ -15V

0 = +3V ~ +15V (150 feet)

- Signals definitions (9 pins D type Connectors)

– DTE / DCE

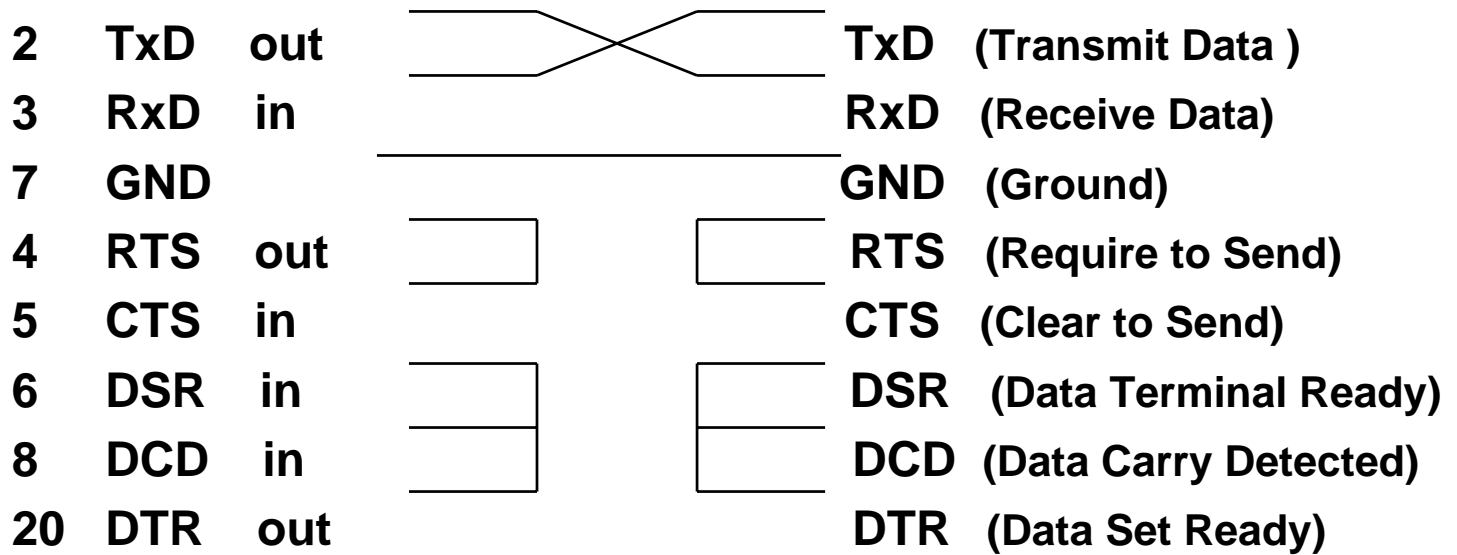


Asynchronous Serial Communication

RS - 232C

- Only for ASCII
- 3 wires connection : (25 pins D type Connector)

DTE / DCE



Ready always, or XON/XOFF protocol, ASCII only!

Asynchronous Serial Communication


RS - 232C

DTE (9 pins connector) in PC

1 protection ground

2 RxD in  RxD (Receive Data)

3 TxD out TxD (Transmit Data)

5 GND  GND (Ground)

6 DSR in DSR (Data Terminal Ready)

7 RTS out RTS (Require to Send)

8 CTS in CTS (Clear to Send)

9 DCD in DCD (Data Carrier Detected)

4 DTR out DTR (Data Set Ready)

No Handshake, should use XON/XOFF protocol, ASCII only

SCI Based Other Communication Protocol

- **Modem**
- **20 mA current loop**
- **RS-485**
- **Local Intercommunication Network (LIN)**
- **Similar method to access other Modules like LIN, SPI, CAN, J1850, Ethernet.....**

S12 SCI Module Registers

SCI0BDH	EQU	\$C8	; Baud Rate Register High
SCI0BDL	EQU	\$C9	; Low
SCI0CR1	EQU	\$CA	; Control Register1
SCI0CR2	EQU	\$CB	; SCI Control Register 2
SCI0SR1	EQU	\$CC	; Status Register 1
SCI0SR2	EQU	\$CD	; Status Register 2(SCISR2
SCI0DRH	EQU	\$CE	; Data Register High
SCI0DRL	EQU	\$CF	; SCI Data Register Low

SCI Initialization

Setup communication protocol:

- Enable T & R in SCICR2
- Write to Baud Rate Register Initial the baud rate,
- Set Enable SCI bit in SCIRC1 Register, Use Default:
8bits, stop bits, Parity Check...

Diff. Form MCU to MCU

SCI Control Register 1 : SCCR1

\$CA	LOOPS	SCISWAI	RSRC	M	WARE	ILT	PE	PT
Reset	0	0	0	0	0	0	0	0

LOOPS=0 **Do Not in Self Loop Mode**

SCISWAI=0 **SCI not stopped in wait mode**

RSRC =0 **Receiver Source not from LOOP mode**

M = 0 **8 Bits, not 9 Bits**

WARE=0 **Wake up Mode IS Idle, Not Address**

ILT =0 **Idle Counter form Start Bits**

PE = 0 **No Parity**

PT = 0 **If Parity, Use Even**

Nice to Use Default !

SCI Control Register 2 : SCCR2

	B7	6	5	4	3	2	1	0
\$CB	SCTIE	TCIE	SCRIE	IDLE	TE	RE	RWU	SBK

SCTIE=0 **Transmit Buffer Empty IRQ Disabled**

TCIE =0 **Transmit Complete IRQ Disabled**

SCRIE=0 **Receiver Buffer Full IRQ Disabled**

IDLE =0 **Idle Line IRQ Disabled**

TE = 0 **Transmit Disabled**

RE = 0 **Receive Disabled**

RWU =0 **Receiver Is Not in Receiver Wake Up Mode**

SBK = 0 **Transmit Break Bit, Not Break.**

Enable SCI0 T & R

```
LDAA # %00001100 ; Initial SCI  
STAA SCI0CR2      ; Enable T & R
```

OR you can use:

```
BSET SCI0CR2, # %00001100 ;Enable T&R
```

Try if it works

SCI Baud Rate Reg. SCIBD

- SCIBD is a 16bit Reg. which has 13 Valid bits,
So, Valid data = $1 \sim 2^{13}-1 = 1 \sim 8191$
- $9600 \times 16 = 153600$
- $24M / 153600 = 156.25$
- $\$9C = 16 \times 9 + 12 = 156$

LDAA #\$9C ; Set Baud Rate = 9600,
STAA SCI0BDL ; \$4E for 19200...

Discussion:

0 Not Valid!

Maximum baud Rate = ?

SCI Initialization

Already did in the debug

SCI0INIT LDAA #0C ; Initial SCI

STAA SCI0CR2 ; Enable T & R

LDAA #\$9C ; Set Baud Rate to 9600

STAA SCI0BDL ; \$4E for 19200...

CLR SCI0CR1 ; 8bit, No Parity.....Not necessary

Transmit & Receive with SCI

SCI States Register (\$CD)

	B7	6	5	4	3	2	1	0
\$CD	TDRE	TC	RDRF	IDLE	OR	NF	FE	PE

TDRE=1,

Transmit Data Reg. **Empty**

TC

Transmit Complete

RDRF=1,

Receive Data Reg. **Full**

IDLE

Line Idle

OR

Over Run Error

NF

Noise Error Flag

FE

Frame Error

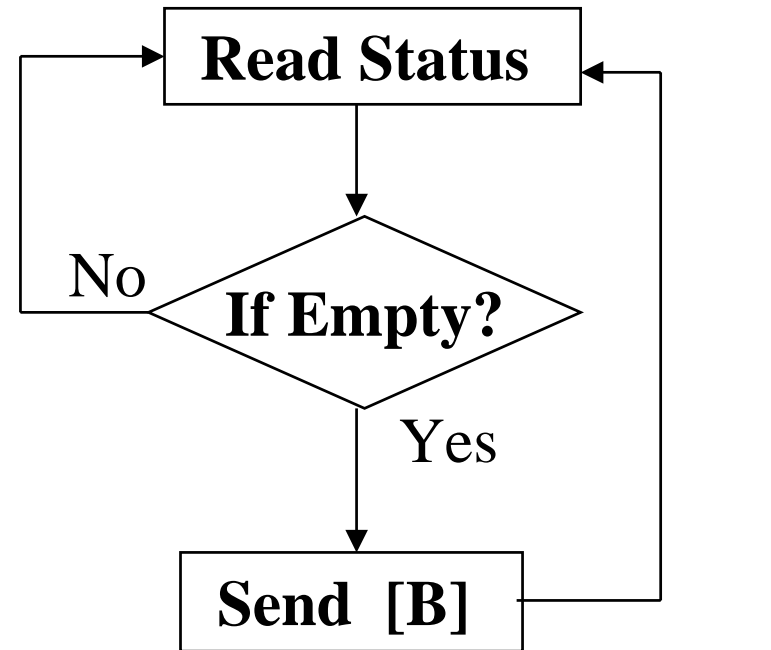
PE

Parity Error

Transmit

Routine OUTCH sends an ASCII in B to SCI0

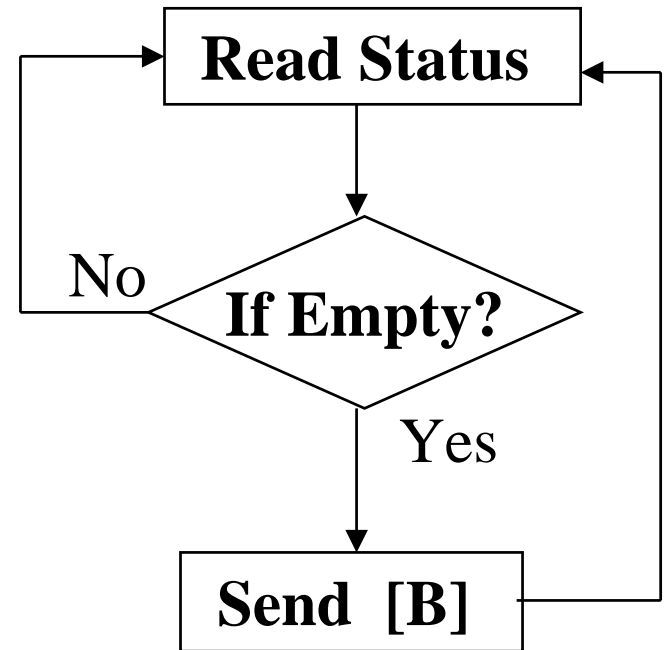
	ORG	\$2000
OUTCH	LDA	#\$80
CHKSTS	BITA	SCI0SR1
	BEQ	CHKSTS
	STAB	SCI0DRL
	RTS	



; TDRE is at bit7
; SCI0 Status Register
; If bit clear, check again
; Write Transmit Data to

Transmit

Routine OUTCH send an
ASCII in B to SCI0



ORG

\$2000

OUTCH

BRCLR

SCI0SR1,#\$80,*

STAB

SCI0DRL

;Write Transmit Data to

RTS

Use Macro in C :

```
#define get_char() {_asm BRCLR $CC,#$80,*; _asm STAB $CF ; }
```

OUTCH

ORG

\$2000

OUTCH
CHKSTS

LDA
BITA
BEQ
STAB
RTS

#\$80

SCI0SR1

CHKSTS

SCI0DRL

; TDRE is at bit7

; SCI0 Status Register

; If bit clear, check again

; Write Transmit Data to

Or:

OUTCH

BRCLR
STAB
RTS

SCI0SR1,\$80,*

SCI0DRL

;Write Transmit Data to

Write Out_char in C

```
#define SCI0SR1 0xcc;  
#define SCI0DRL 0xcf;
```

```
Put_Char(char a)  
{ while ((*SCI0SR1 & 0x80) == 0);  
  *SCI0DRL = c;  
}
```

That may be a low lever I/O functions for C

INCH

```
INCH BRCLR SCI0SR1,$20,INCH ;Check if keyboard hit  
LDAB SCI0DRL  
RTS
```

```
#define SCI0SR1 0xcc;  
#define SCI0DRL 0xcf;
```

```
char get_char();  
{ while ((*SCI0SR1 & 0x20) == 0);  
    return(*SCI0DRL); }
```

Use Macro:

```
#define get_char() {_asm BRCLR $CC,$20,*; _asm LDAB $CF ; }
```

Problems of Polling in I/O status

- It keeps 100% CPU time :

For INCH:

BRCLR SCI0SR1,\$20,* ; 4 Cycles

When CPU bus clock = 24M,
CPU run the instruction 6,000,000 times / Sec!

- For OUTCH, 9600b/s, ~ 1mS/char

SCI Interrupts

There are 3 Interrupt sources in SCI:

- **Receive IRQ**
 - Receiver Register Full Interrupt
- **Transmit IRQ**
 - Transmit Register Empty Interrupt
 - » **Next Char can be written**
- **Transmit Completed Interrupt**
 - » Drive one wire to three status or Receive
- **Before Enable IRQ Remember to Initial Vectors !**
- **Because the Monitor, Vectors moved !**

SCI Control Register 2 : SCCR2

	B7	6	5	4	3	2	1	0
\$CB	SCTIE	TCIE	SCRIE	IDLE	TE	RE	RWU	SBK

SCTIE=1 **Transmit Buffer Empty IRQ Enabled**

TCIE =0 **Transmit Complete IRQ Disabled**

SCRIE=1 **Receiver Buffer Full IRQ Enabled**

IDLE =0 **Idle Line IRQ Disabled**

TE = 0 **Transmit Disabled**

RE = 0 **Receive Disabled**

RWU =0 **Receiver Is Not in Receiver Wake Up Mode**

SBK = 0 **Transmit Break Bit, Not Break.**

Enable the SCI Interrupt

- Write to SCCR2 Interrupt control Register to enable the IRQ:

	B7	6	5	4	3	2	1	0
\$CB	SCTIE	TCIE	SCRIE	IDLE	TE	RE	RWU	SBK

- Initial the Victors before the enable!
- Use CLI to Enable IRQ!

System Interrupt Vector Table 1

Vector Address	Interrupt Source	CCR Mask	Local Enable	HPRIO Value to Elevate
FFFFE, \$FFFF	Reset	None	None	—
FFFC, \$FFFD	Clock Monitor fail reset	None	COPCTL (CME, FCME)	—
FFFA, \$FFFB	COP failure reset	None	COP rate select	—
FFF8, \$FFF9	Unimplemented instruction trap	None	None	—
FFF6, \$FFF7	SWI	None	None	—
FFF4, \$FFF5	XIRQ	X-Bit	None	—
FFF2, \$FFF3	IRQ	I-Bit	INTCR (IRQEN)	\$F2
FFF0, \$FFF1	Real Time Interrupt	I-Bit	CRGINT (RTIE)	\$F0
FFEE, \$FFEF	Timer channel 0	I-Bit	TMSK1 (C0I)	\$EE
FFEC, \$FFED	Timer channel 1	I-Bit	TMSK1 (C1I)	\$EC
FFEA, \$FFEB	Timer channel 2	I-Bit	TMSK1 (C2I)	\$EA
FFE8, \$FFE9	Timer channel 3	I-Bit	TMSK1 (C3I)	\$E8
FFE6, \$FFE7	Timer channel 4	I-Bit	TMSK1 (C4I)	\$E6

System Interrupt Vector Table 2

FFFE4, \$FFE5	Timer channel 5	I-Bit	TMSK1 (C5I)	\$E4
FFFE2, \$FFE3	Timer channel 6	I-Bit	TMSK1 (C6I)	\$E2
FFFE0, \$FFE1	Timer channel 7	I-Bit	TMSK1 (C7I)	\$E0
FFDE, \$FFDF	Timer overflow	I-Bit	TMSK2 (TOI)	\$DE
FFDC, \$FFDD	Pulse accumulator A overflow	I-Bit	PACTL (PAOVI)	\$DC
FFDA, \$FFDB	Pulse accumulator input edge	I-Bit	PACTL (PAI)	\$DA
FFD8, \$FFD9	SPI0	I-Bit	SP0CR1 (SPIE, SPTIE)	\$D8
FFD6, \$FFD7	SCI 0	I-Bit	SC0CR2 (TIE, TCIE, RIE, ILIE)	\$D6
FFD4, \$FFD5	SCI 1	I-Bit	SC1CR2 (TIE, TCIE, RIE, ILIE)	\$D4
FFD2, \$FFD3	ATD0	I-Bit	ATD0CTL2 (ASCIE)	\$D2
FFD0, \$FFD1	ATD1	I-Bit	ATD1CTL2 (ASCIE)	\$D0
FFCE, \$FFCF	Port J	I-Bit	PTJIF (PTJIE)	\$CE
FFCC, \$FFCD	Port H	I-Bit	PTHIF (PTHIE)	\$CC
FFCA, \$FFCB	Modulus Down Counter underflow	I-Bit	MCCTL (MCZI)	\$CA

Debug: Move Vectors to User Area

```
V38          LDX    $EFD6
              PSHX
              RTS

*
              ORG    $FF8C ; Vectors table
*
FF8C          FDB    V1      ; PWM Emergency Shutdown
              FDB    V2      ; Port P Interrupt
              FDB    V3      ; MSCAN 4 transmit
              .....
              FDB    V37     ; SCI 1
FFD6          FDB    V38     ; SCI 0
              FDB    V39     ; SPI0
              .....
              FDB    V56     ; COP failure reset
              FDB    V57     ; Clock Monitor fail reset
FFFE          FDB    V58     ; Reset
```

SCI Receiver Interrupt Service Routine

SCI0SR1 **EQU** **\$CC** ; Status Reg.

SCI0DRL **EQU** **\$CF** ; Data Reg.

SCI0ISR **LDAA** **SCI0SR1** ; Clear IRQ !!!

LDAB **SCI0DRL** ; Get the Char.

Put the char. Somewhere

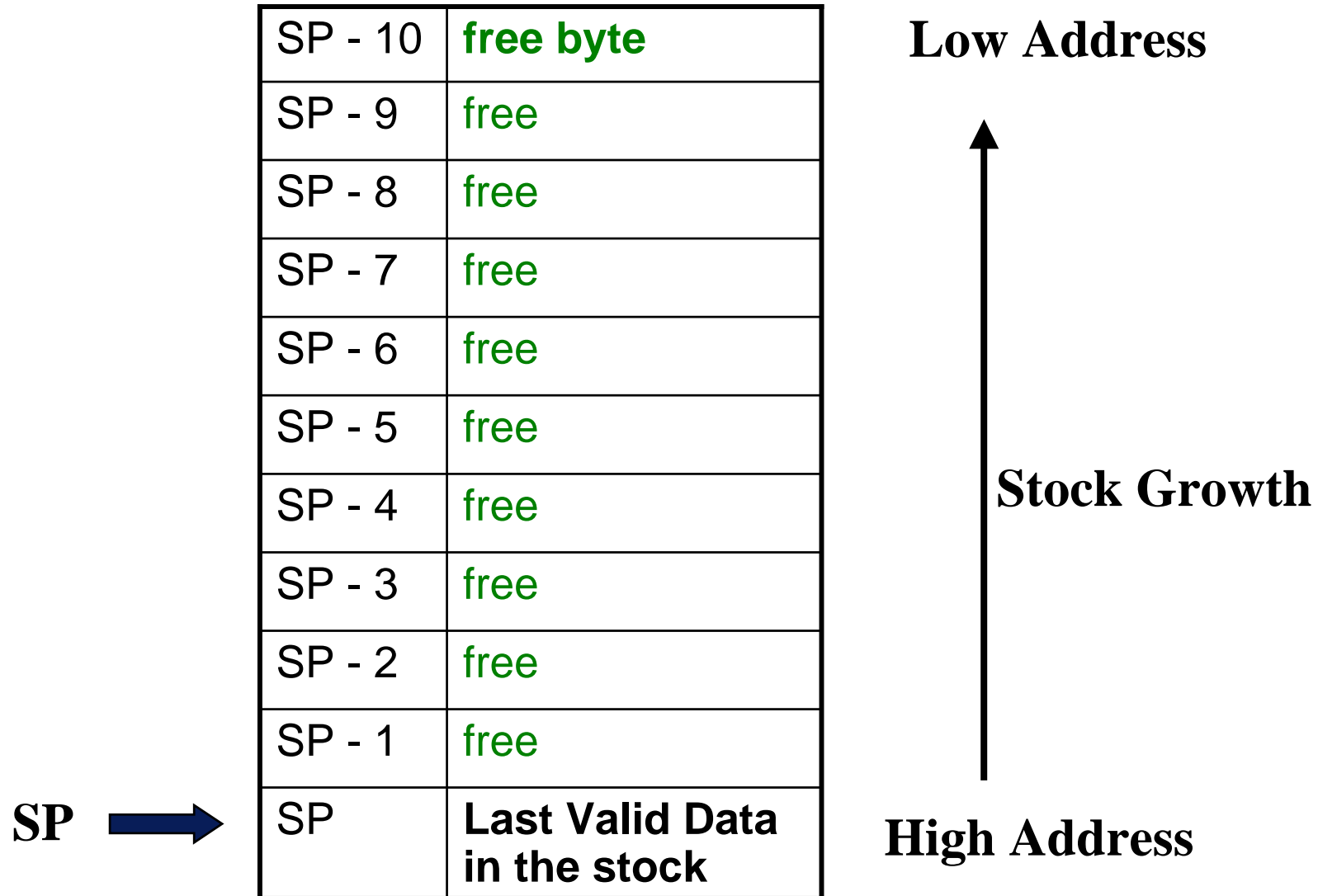
.....

RTI

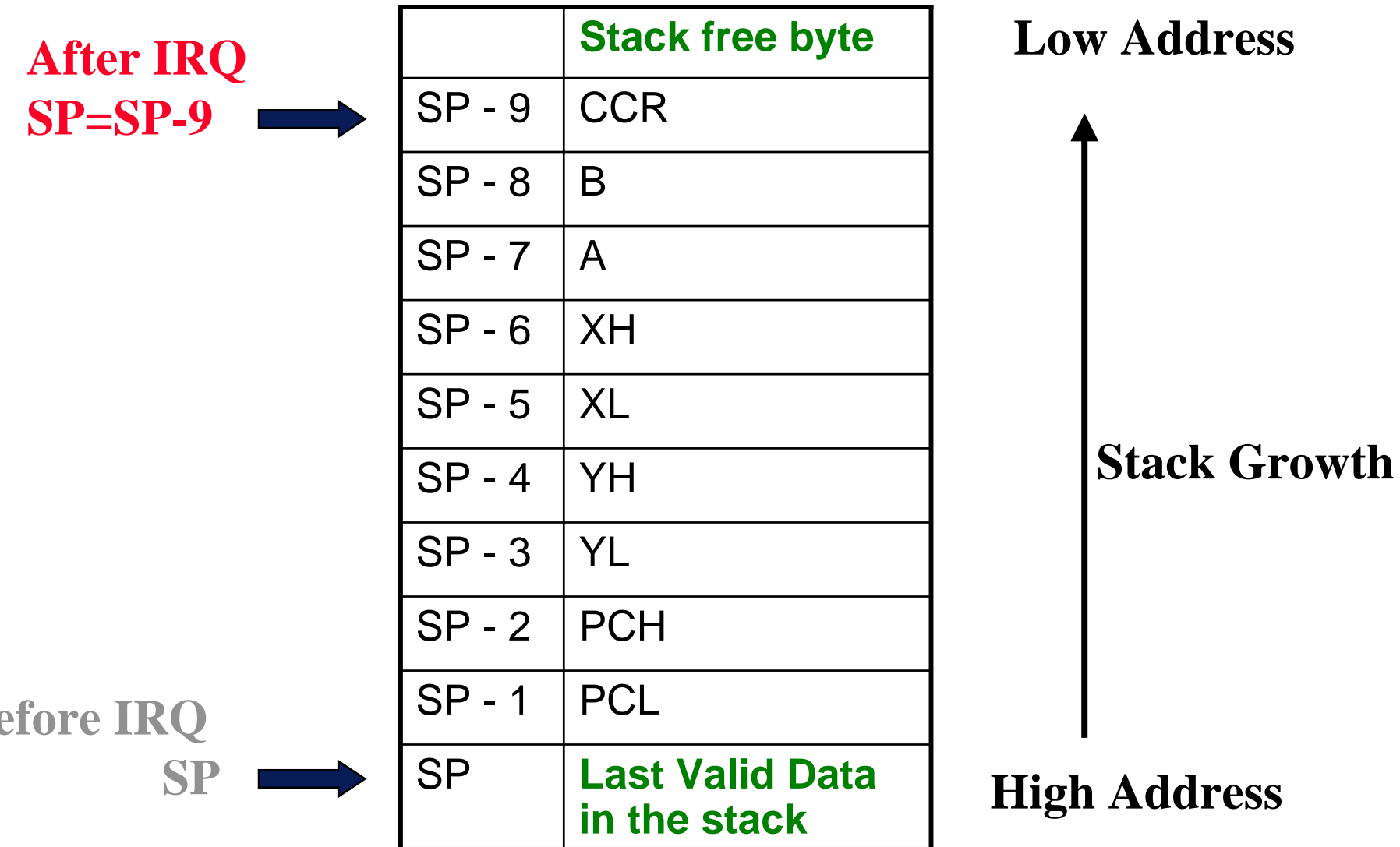
ORG **\$EFD6**

DC.W **SCI0ISR**

Stack Structure before IRQ



Stack Structure after IRQ Response



Stack Structure after RTI

SP - 10	free byte
SP - 9	free
SP - 8	free
SP - 7	free
SP - 6	free
SP - 5	free
SP - 4	free
SP - 3	free
SP - 2	free
SP - 1	free
SP	Last Valid Data in the stock

Low Address



Stock Growth

High Address

After RTI
 $SP = SP + 9$



Summery of SCI Receive IRQ

- Initial SCI (Monitor did already for you)
- Write Interrupt service Routine
 - **Clear IRQ Flag** Clear RDRF by reading SCI status register 1 (SCISR1) with RDRF set and then reading SCI data register low (SCIDRL).
 - **Get the Character**
 - **RTI**
- Initial INT Victor Table (User's victor Table)
- Enable SCI Receive Interrupt
- Enable System IRQ in CCR by CLI

Questions: How to Do the Transmit Interrupt?

Reference book:

“Embedded System Building Block”

Jean J. Labrosse

How to write device driver?

What is IP?