Program Interrupts

Chapter 12

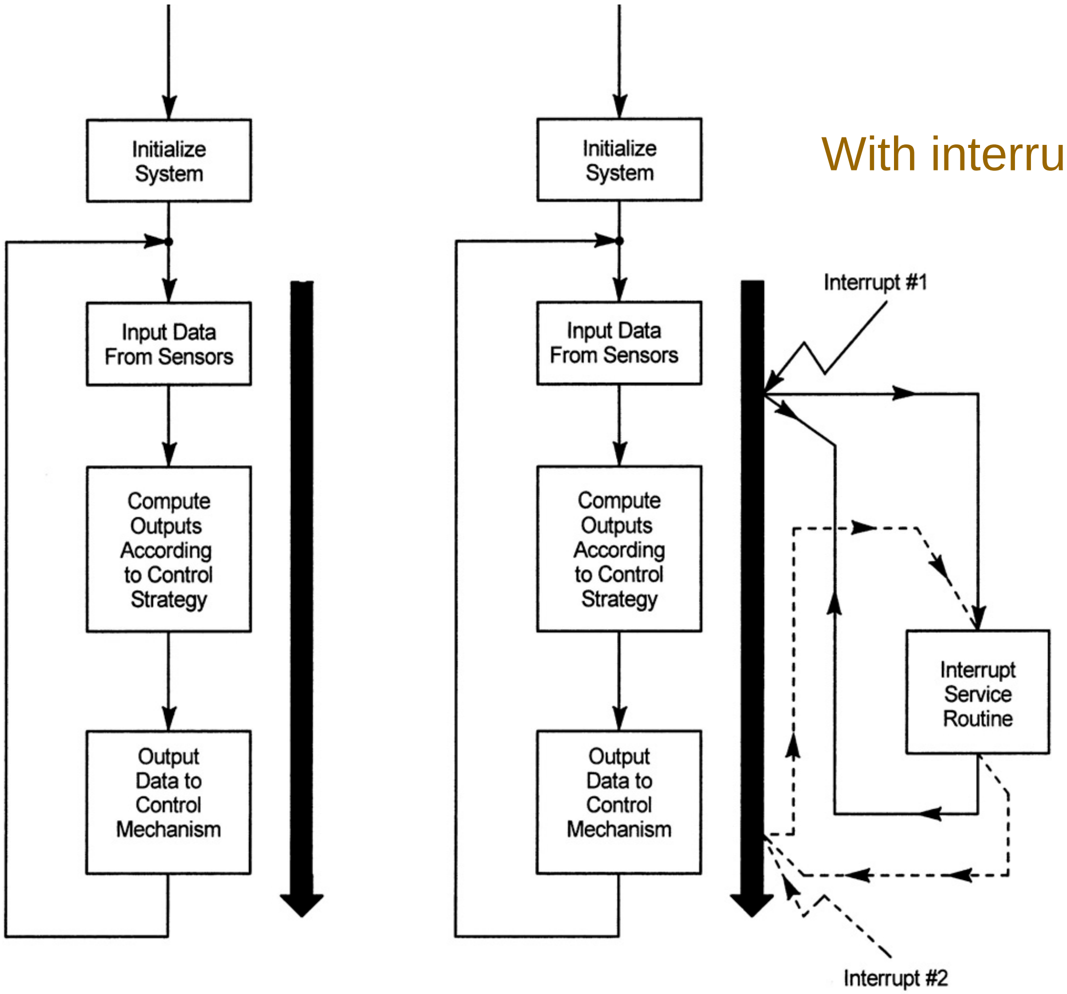
Program interrupts

- An “event” interrupts a running program to force execution of another program
  - Signal from an external device
  - Signal from an internal device
  - Exceptional condition (ex. divide by 0)
  - Software interrupt instruction (swi)
- CPU executes an “interrupt service routine” to deal with the interrupt event

Process control SW (Fig. 12-1)

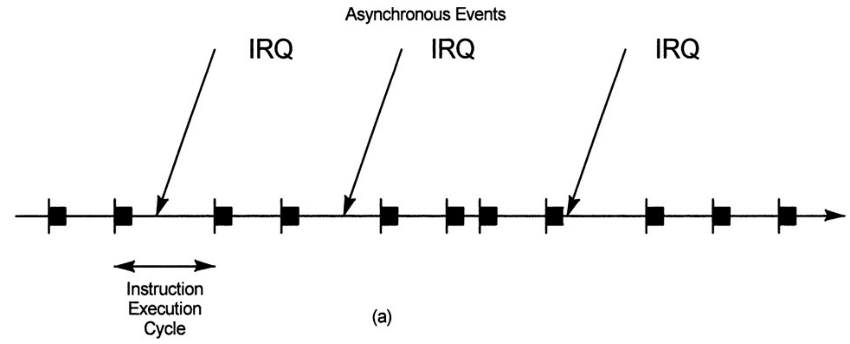
Typical

With interrupts

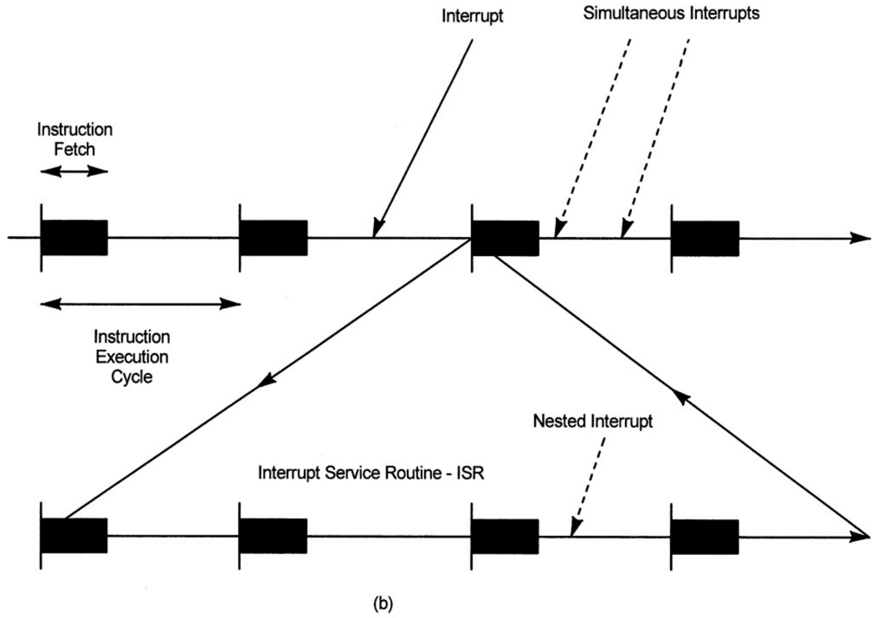


Asynchronous events (Fig. 12-2)

Timeline:



Expanded Timeline:



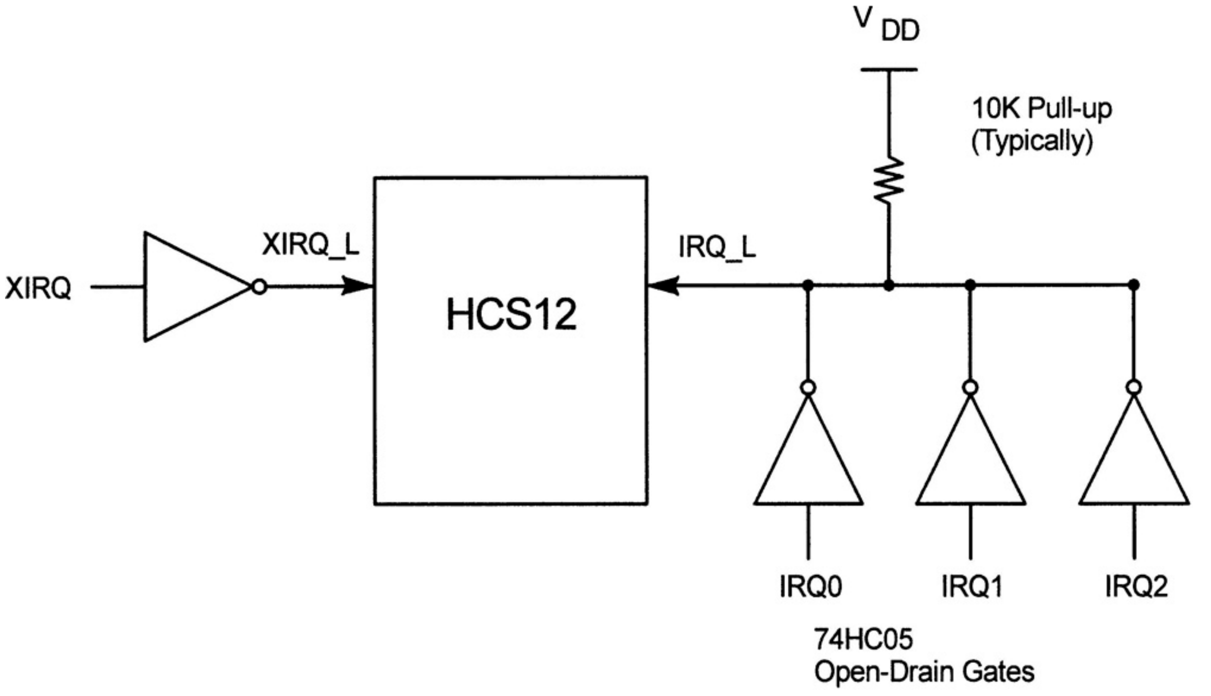
HCS12 interrupt sources

- Signals from external devices on designated CPU pins

  $\overline{\text{IRQ}}$ (interrupt request) pin

  $\overline{\text{XIRQ}}$ (non-maskable interrupt request) pin
- Signals from internal functions
  - Timers
  - Communication channels
  - Detected failures (clock, illegal instruction,etc.)
- Software interrupt instruction (SWI)

Hardware interface for IQR* and XIRQ* (Figure 12-5)



Enabling/disabling interrupts

Condition Code Register (CCR)

S	X	H	I	N	Z	V	C

Mask $\overline{\text{XIRQ}}$ Mask $\overline{\text{IRQ}}$

1 “masks” (disables) $\overline{\text{IRQ}}$/$\overline{\text{XIRQ}}$

0 “unmasks” (enables) $\overline{\text{IRQ}}$/$\overline{\text{XIRQ}}$ [default state]

Disble $\overline{\text{IRQ}}$: ORCC #%00010000 (formerly SEI)

Enable $\overline{\text{IRQ}}$: ANDCC #%11101111 (formerly CLI)

Enable XIRQ: ANDCC #%10111111

X bit cannot be set to 1 after it has been cleared.

Therefore, once enabled, XIRQ cannot be disabled.

CPU actions in an interrupt sequence

1. Check interrupt signals <u>after</u> completing current instruction
2. Determine address of interrupt service routine (interrupt vector)
3. Push PC (return address) on stack
4. Push X, Y, A, B, CCR onto stack
5. Set I bit in CCR (mask other interrupts)
6. Branch to interrupt service routine

Return from interrupt

- Execute RTI instruction
  - RTI pulls from stack: CCR, B, A, X, Y, PC
- Pulling PC from stack returns CPU to the interrupted program
- Restoring original CCR also restores original X and I bits (reenables interrupts)
  - Software does not need to reenable interrupts

Interrupt vectors

- “Interrupt vector” = start address of interrupt service routine
- Interrupt vectors for all interrupt sources are stored in the last bytes in memory

Examples:

FFFE,FFFF: Reset vector
FFFC,FFFD: Clock monitor fail (reset)
FFFA,FFFB: COP failure (reset)
FFF8,FFF9: Unimplemented opcode trap
FFF6,FFF7: SWI
FFF4,FFF5: XIRQ pin
FFF2,FFF3: IRQ pin
FF00…FFF1: Built-in functions (device-specific)

Interrupt vector assignments for HCS12

TABLE 12-1 Interrupt Vector Assignments

Priority[a]	Vector Address	Interrupt Source	Local Enable Bit	See Register	See Chapter	HPRIO Value to Promote
—	$FF80:FF89	Reserved	—	—	—	—
58	$FF8A:FF8B	VREG LVI	LVIE	CTRL0	—	$8A
57	$FF8C:FF8D	PWM Emergency Shutdown	PWMIE	PWMSDN	14	$8C
56	$FF8E:FF8F	Port P	PIEP[7:0]	PIEP	—	$8E
33	$FF90:FFAF	Reserved	—	—	—	—
39	$FFB0:FFB1	CAN Transmit	TXEIE[2:0]	CANTIER	16	$B0
38	$FFB2:FFB3	CAN Receive	RXFIE	CANRIER	16	$B2
37	$FFB4:FFB5	CAN Errors	CSCIE, OVRIE	CANRIER	16	$B4
36	$FFB6:FFB7	CAN Wake-up	WUPIE	CANRIER	16	$B6
35	$FFB8:FFB9	Flash	CCIE, CBEIE	FCNFG	—	$B8
—	$FFBA:FFC3	Reserved	—	—	—	—
29	$FFC4:FFC5	CRG Self Clock Mode	SCMIE	CRGINT	20	$C4
28	$FFC6:FFC7	CRG PLL Lock	LOCKIE	CRGINT	20	$C6
—	$FFC8:FFCD	Reserved	—	—	—	—
24	$FFCE:FFCF	Port J	PIEJ[7:6]	PIEJ	11	$CE
—	$FFD0:FFD1	Reserved	—	—	—	—
22	$FFD2:FFD3	A/D Converter	ASCIE	ATDCTL2	17	$D2
—	$FFD4:FFD5	Reserved	—	—	—	—

Continued on next slide

Interrupt vector assignments for HCS12

Continued from previous slide

TABLE 12-1 Continued

Priority[a]	Vector Address	Interrupt Source	Local Enable Bit	See Register	See Chapter	HPRIO Value to Promote
20	$FFD6:FFD7	SCI Serial System	TIE, TCIE, RIE, ILIE	SCICR2	15	$D6
19	$FFD8:FFD9	SPI Serial Peripheral System	SPIE, SPTIE	SPICR1	15	$D8
18	$FFDA:FFDB	Pulse Accumulator Input Edge	PAI	PACTL	14	$DA
17	$FFDC:FFDD	Pulse Accumulator Overflow	PAOVI	PACTL	14	$DC
16	$FFDE:FFDF	Timer Overflow	TOI	TSCR2	14	$DE
15	$FFE0:FFE1	Timer Channel 7	C7I	TIE	14	$E0
14	$FFE2:FFE3	Timer Channel 6	C6I	TIE	14	$E2
13	$FFE4:FFE5	Timer Channel 5	C5I	TIE	14	$E4
12	$FFE6:FFE7	Timer Channel 4	C4I	TIE	14	$E6
11	$FFE8:FFE9	Timer Channel 3	C3I	TIE	14	$E8
10	$FFEA:FFEB	Timer Channel 2	C2I	TIE	14	$EA
9	$FFEC:FFED	Timer Channel 1	C1I	TIE	14	$EC
8	$FFEE:FFEF	Timer Channel 0	C0I	TIE	14	$EE
7	$FFF0:FFF1	Real-Time Interrupt	RTIE	CRGINT	14	$F0
6	$FFF2:FFF3	IRQ_L Pin	IRQEN	INTCR	12	$F2
5	$FFF4:FFF5	XIRQ_L Pin	X bit	CCR	12	—
4	$FFF6:FFF7	SWI	None	—	—	—
3	$FFF8:FFF9	Unimplemented Instruction Trap	None	—	—	—
2	$FFFA:FFFB	COP Failure Reset	CR2:CR1:CR0	COPCTL	—	—
1	$FFFC:FFFD	Clock Monitor Fail Reset	CME, SCME	PLLCTL	—	—
0	$FFFE:FFFF	External Reset	None	—	—	—

[a] The numbers given for the priority show zero as the highest priority. This numbering scheme is also used by the CodeWarrior linker to locate the vector in the proper place in memory. See Example 12-3.

Creating the reset vector
(Code Warrior sample program)

```
; code section
          ORG   ROMStart
Entry:
          LDS   #RAMEnd+1            ; initialize the stack pointer
          CLI                        ; enable interrupts
mainLoop:
            …….


;**************************************************************
;*                Interrupt Vectors                            *
;**************************************************************
          ORG   $FFFE        ; Address for storing reset vector
          DC.W  Entry        ; Reset Vector
```

Initializing HCS12 Interrupt Vectors

```
; Initialize Timer Channel 0 Interupt vector
TC0             EQU     $FFEE   ; Address of the vector
; Put in the main program
        ORG     ROMstart
Entry:  (main program)
; The interrupt service routine starts with a label
; at the first instruction to be executed
TC0ISR:
        Timer ISR Instructions
        rti     ; The isr ends with a RTI
; Locate the vectors
        ORG     TC0
        DC.W    TC0ISR ; The label is the address of the ISR
        ORG     $FFFE  ; Address of reset vector
        DC.W    PROG
```

Nonmaskable interrupt priorities

TABLE 12-2 Nonmaskable Interrupt Priorities

Priority	Nonmaskable Interrupt Source	Vector Address	Enable Bit	See Register
5	XIRQ_L	$FFF4:FFF5	X	CCR
4	Software Interrupt Instruction (SWI)	$FFF6:FFF7	None	None
3	Unimplemented Opcode Trap	$FFF8:FFF9	None	None
2	COP Reset	$FFFA:FFFB	CR2, CR1, CR0	COPCTL
1	Clock Monitor Reset	$FFFC:FFFD	CME, SCME	PLLCTL
0	System Reset (RESET_L)	$FFFE:FFFF	None	None

Highest priority interrupt register (HPRIO)

PSEL7	PSEL6	PSEL5	PSEL4	PSEL3	PSEL2	PSEL1	0

- To dynamically change interrupt priorities of IRQ and lower-priority interrupts
- Promote any one interrupt to “highest priority”
- Write 7-bit code (from Table 12-1) to HPRIO
- Should be done with CCR I bit set

Example

Write a small segment of code to raise Timer Channel 2 to the highest priority position.

```
hprio1c.asm            Assembled with CASM   05/28/1998  02:20  PAGE 1

 0000              1  HPRIO:    EQU     $1F              ; HPRIO address
 0000              2  TC2VECT:  EQU     $FFEA            ; Channel 2 Vector
                   3  ; . . .
                   4  ; Mask interrupts while setting HPRIO
 0000 1410         5            sei                      ; Set I-bit
                   6  ; Raise Timer Channel 2 to the highest priority
 0002 CCFFEA       7            ldd     #TC2VECT
 0005 5B1F         8            stab    HPRIO
 0007 10EF         9            cli                      ; Clear interrupt mask
```

HCS12 interrupt system registers

TABLE 12-4 Interrupt System Registers

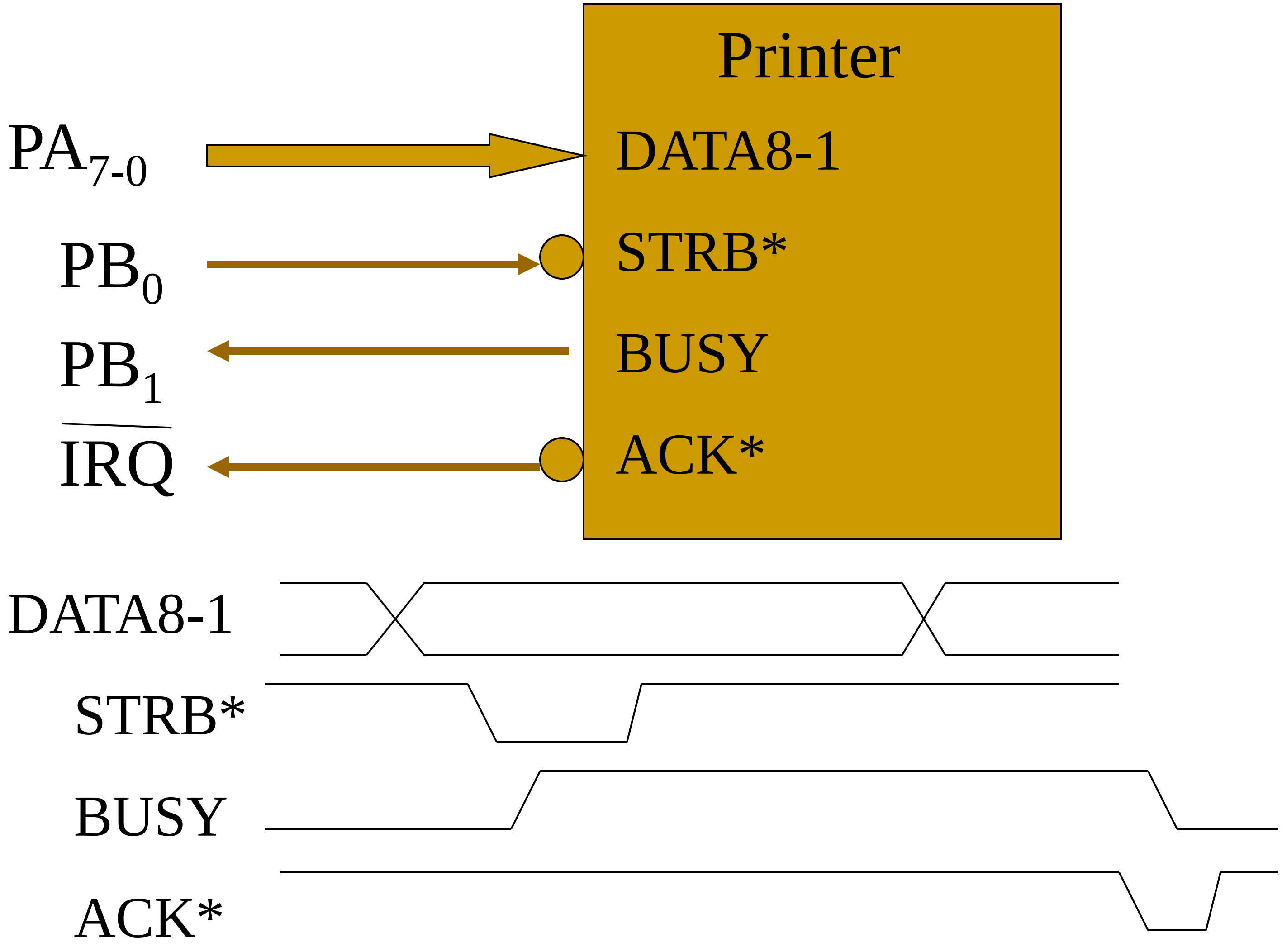
Name	Register	Address Base+
ARMCOP	Arm COP Register	$003F
COPCTL	COP Control Register	$003C
HPRIO	Highest Priority Interrupt Register	$001F
INTCR	Interrupt Control Register	$001E
ITCR	Interrupt Test Control Register	$0015
ITEST	Interrupt Test Registers	$0016
PERJ	Pull-up or Pull-down Enable Port J	$026C
PERP	Pull-up or Pull-down Enable Port P	$025C
PIEJ	Port J Interrupt Enable Register	$026E
PIEP	Port P Interrupt Enable Register	$025E
PIFJ	Port J Interrupt Flag Register	$026F
PIFP	Port P Interrupt Flag Register	$025F
PLLCTL	CRG PLL Control Register	$003A
PPSJ	Port J Polarity Select	$026D
PPSP	Port P Polarity Select	$025D

Interrupt control register (INTCR)

	7	6	5	4	3	2	1	0
	IRQE	IRQEN	DLY	0	0	0	0	0
RESET Values:	0	1	1	0	0	0	0	0

- IRQE ($\overline{\text{IRQ}}$ edge-sensitivity)
  - 1 => $\overline{\text{IRQ}}$ responds only to falling edges
  - 0 => $\overline{\text{IRQ}}$ responds only to low levels
- IRQEN ($\overline{\text{IRQ}}$ enable)
  - 1 => $\overline{\text{IRQ}}$ pin connected to interrupt logic
  - 0 => $\overline{\text{IRQ}}$ pin disconnected from interrupt logic
- DLY : oscillator startup delay on exiting “stop mode”
  - 1 => delay
  - 0 => no delay

Example: Interrupt-driven printing



Program-controlled solution (no interrupt)

```
        ldx     #string
Loop:   ldaa    1,x+            ;get next character
        beq     Return ;quit on NULL
        staa    PORTA           ;character to printer
        bclr    PORTB,1         ;force STB* to 0
        bset    PORTB,1         ;force STB* to 1 (pulse)
Wait:   brset   PORTB,2,Wait  ;wait until BUSY=0
        bra     Loop
Return: rts
```

Note wasted time waiting for BUSY.

Interrupt-driven solution

```
;Printer ISR – Send next character to printer
;   Saved_Pointer has address of next character
PrintISR:   ldx    Saved_Pointer
            ldaa   1,x+                 ;get next character
            beq    Return      ;quit on NULL
            stx    Saved_Pointer  ;save for next interrupt
            staa   PORTA                ;character to printer
            bclr   PORTB,1              ;force STB* to 0
            bset   PORTB,1              ;force STB* to 1
Return:     rti
```

Main program setup

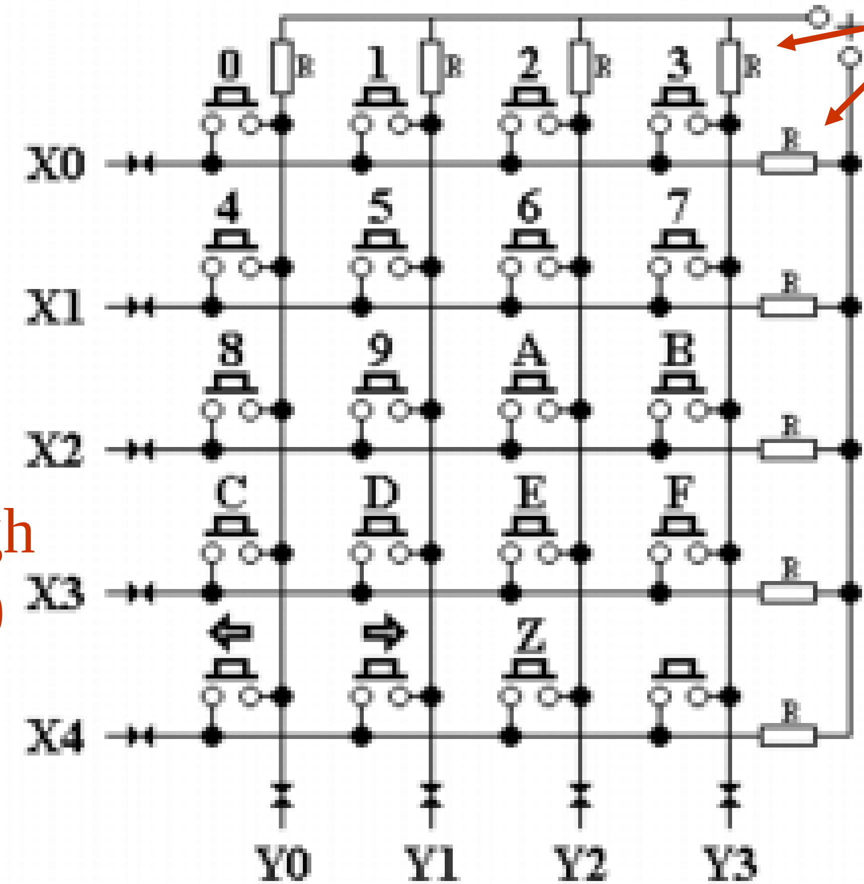
```
Saved_Pointer      ds.w       1
String             dc.b        “This is a string”,0
; Set up ports and pointer
Entry:     bset     DDRA,$ff;PortA=output
           bset     DDRB,%01            ;PB0=output
           bclr     DDRB,%10            ;PB1=input
           ldx      #String             ; address of String
           stx      Saved_Pointer       ; save for ISR
           cli                          ; enable interrupt
           swi                          ; print 1st character
;******  rest of the program
;
           org      $fff2
           dc.w     PrintISR            ; IRQ vector
           org      $fff6
           dc.w     PrintISR            ; SWI vector
           org      $fffe
           org      Entry               ; reset vector
```

Interrupting keyboard component (IT Keyboard)

Pullup Resistors

Row Lines

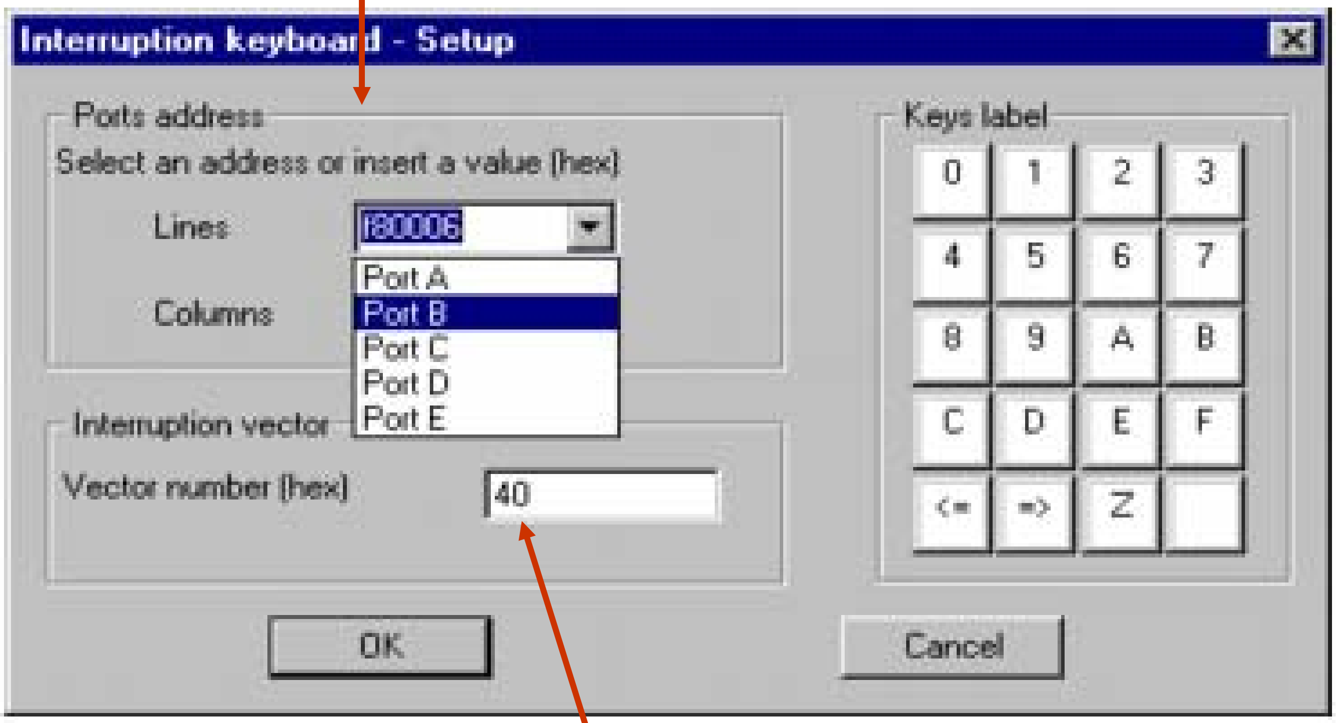
(read through a port)



Column Lines (drive with a port)

IT_Keyboard Setup

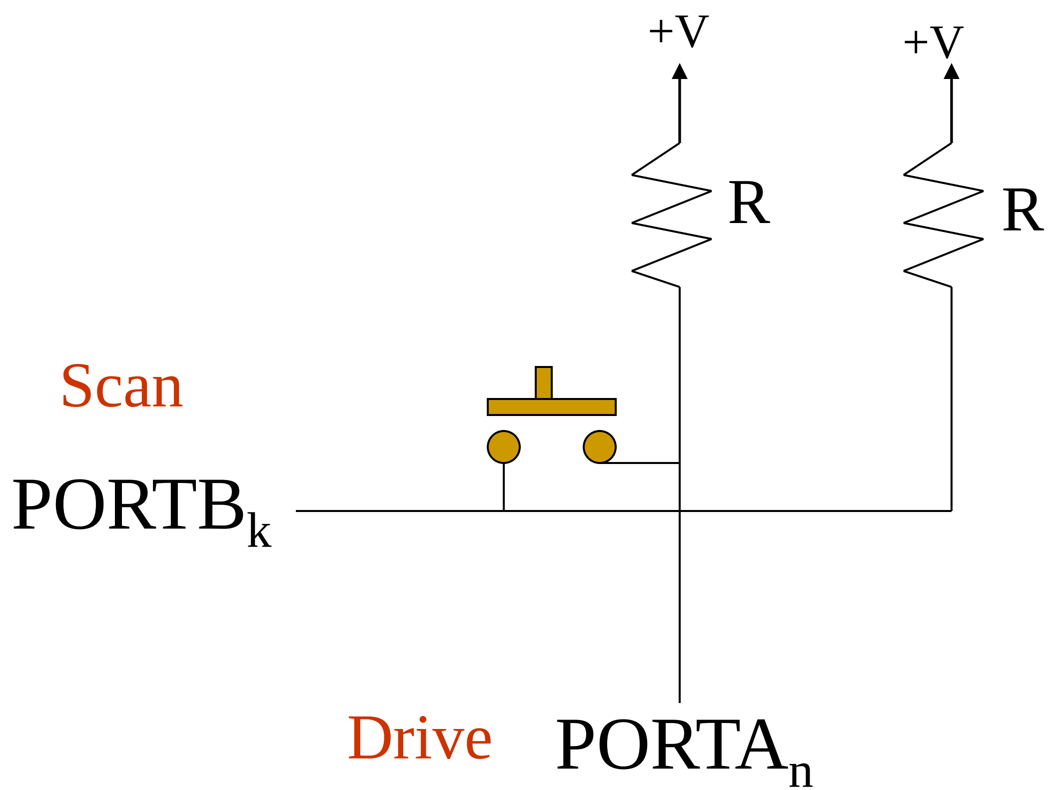
Assign ports (lines/columns connect to low bits of port)



Keys may be relabelled

IRQ is vector #6

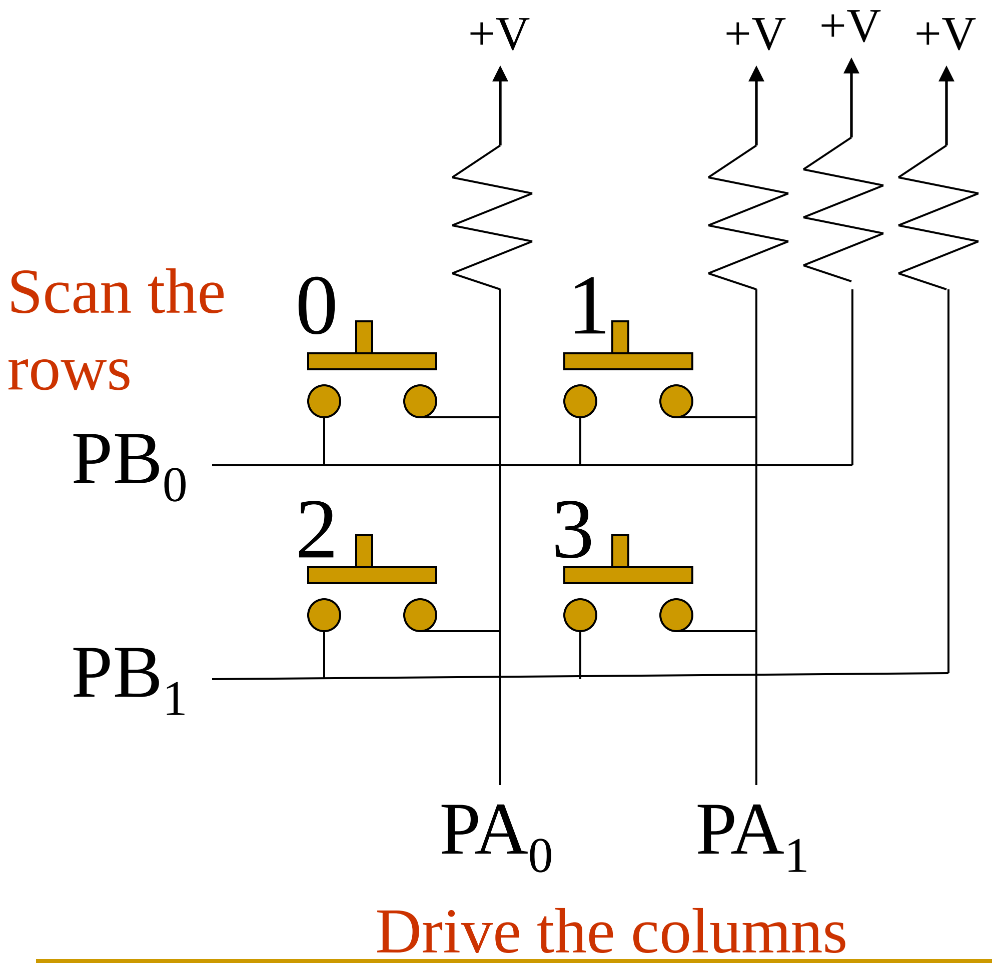
“Scanning” the keyboard



PA_k out	Push Button	PB_n in
1	up	1
1	dn	1
0	up	1
0	dn	0*

- Detect pressed switch by forcing column line to 0 and read row line = 0 (“activates” the column)
- Can interchange roles of row and column lines.

"Scanning" the keyboard



Drive PA0-1	Push Button	Read PB0-1
11	0,1,2,3	11
10	0	01
10	1	10
10	2 or 3	11
01	0 or 1	11
01	2	01
01	3	10
00	0 or 2	01
00	1 or 3	10

Keyboard scan algorithm

- Force one column K = 0 and others = 1
- Examine each row L, to test key at intersection of row L and column K
  - Row L = 1 (no key press at K:L)
  - Row L = 0 (key press at K:L)
- If no keys pressed in column K, force next column to 0 with others = 1, etc.
  - Exactly one column=0 at any time

Interrupting keyboard

- Interrupt generated when:
  1. Column C activated (forced to 0)
  2. Any key in row C is pressed
- If all columns activated (all forced to 0), any key press generates an interrupt
- Interrupt-driven keyboard scan:
  - Activate all columns
  - A key press triggers an interrupt
  - Perform regular scan algorithm to find the key