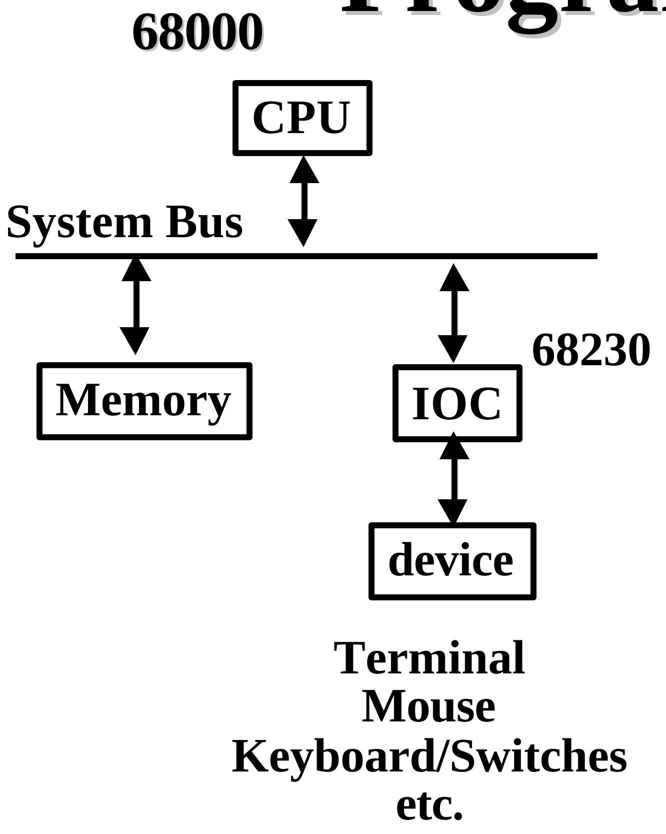
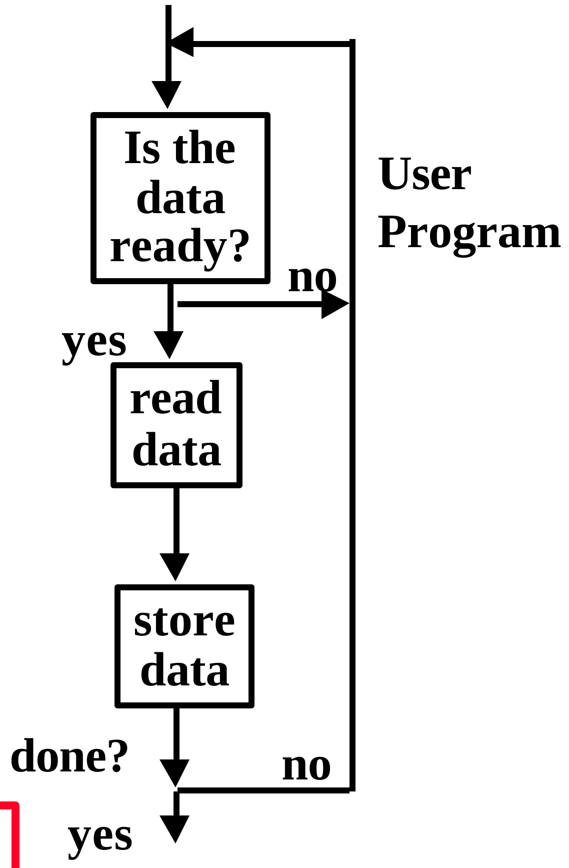
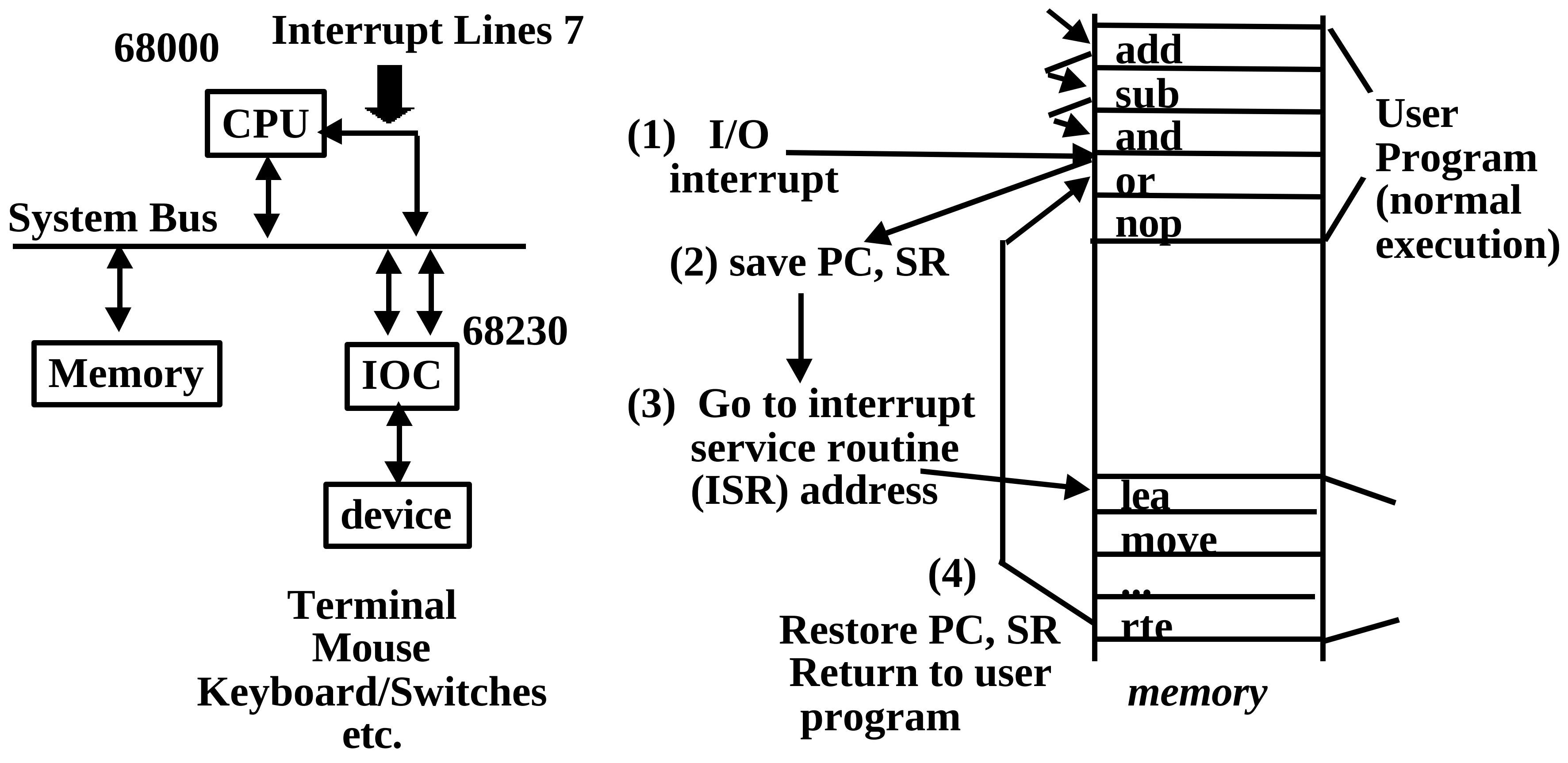
Programmed I/O (Polling)





- A busy-wait loop is used in this I/O method
- Not an efficient way to use the CPU unless the I/O device is very fast (faster than the CPU).
- But checks for I/O completion can be dispersed among useful computation

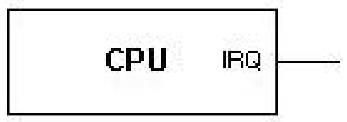
Interrupt-Driven Data Transfer



- User program normal execution only halted during actual data transfer
- More efficient than polling I/O

Interrupts & Exceptions

- **Conditions interrupting ordinary program execution are called exceptions when caused by software sources internal to the CPU.**
- **Interrupts are exceptions which are caused by hardware sources external to the CPU.**
- **An interrupt or exception generally requires special handling by the CPU in terms of executing an Interrupt Service Routine (ISR).**
- **Example: An I/O device informs the CPU that data is ready and requests special processing by setting an Interrupt Request line (IRQ) to True.**
  - **68000 has 7 such IRQ lines: (IRQ1-IRQ7).**



- **If two hardware interrupts occur *simultaneously* the IRQ with a higher priority (higher IRQ number) gets serviced.**
- **An interrupt with a higher IRQ can interrupt the ISR of an interrupt with a lower IRQ.**

Vectored Interrupts

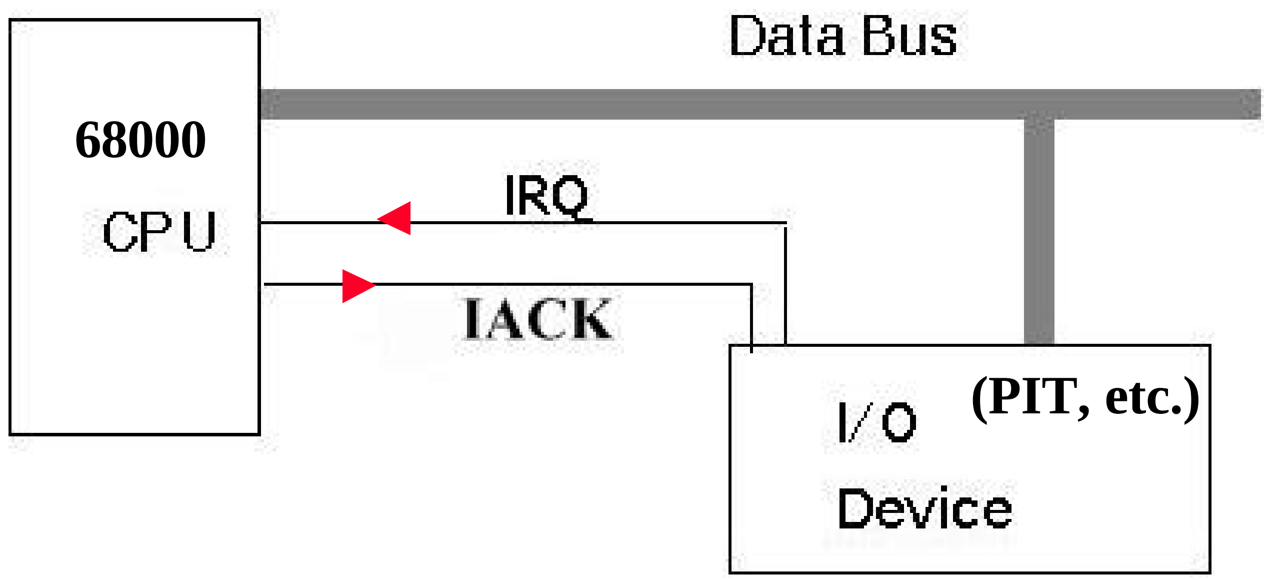
- When the 68000 senses that an Interrupt Request is pending it *stops* the normal program execution and must identify the type of interrupt or exception to execute the correct handling routine.
- The 68000 must be in *supervisor mode* as set in SR (S bit = 1) to handle interrupt routines.
- The 68000 allows 255 such routines and stores their location (called a vector) addresses in the first 1K of 68000 program memory.
  - This area is called the *exception vector table*:
    - vector #1 - SPECIAL - for system start-up
    - vector #2

      …..
    - vector #255

    Vectors 64-255 are user interrupt vectors
  - Address of exception vector = 4 x exception vector number

Vectored Interrupts

- The interrupt vector is provided to the CPU on the data bus by the interrupting I/O device from an interrupt vector register:

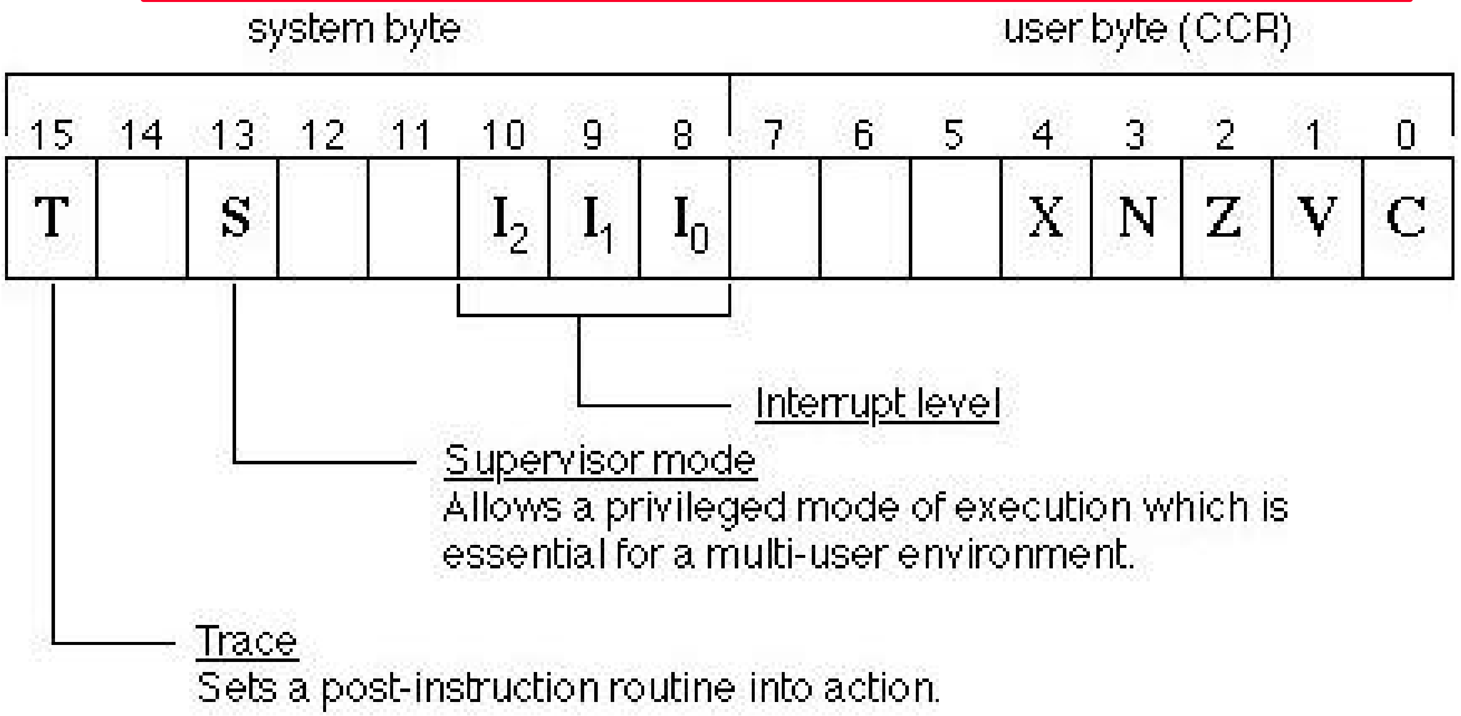


- When the 68000 accepts the interrupt, it acknowledges this to the device using line IACK and it looks for the interrupt (or exception) vector on the data bus.

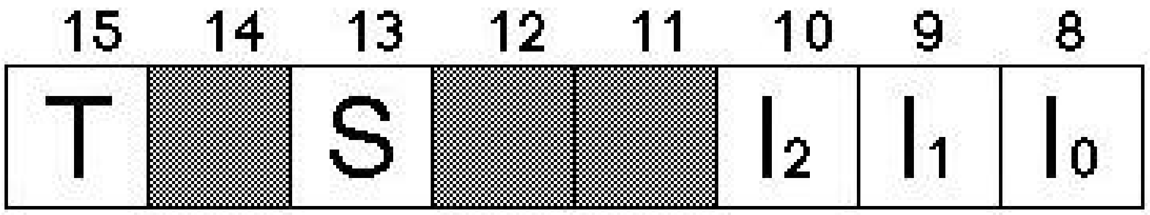
Status Register (SR) & Interrupts

$I_2\ I_1\ I_0$: Interrupt level (interrupt mask bits, value range = 0 to 7)

- **The 68000 only accepts interrupts with a higher level than that set by the interrupt mask bits**
- **An interrupt with level 7 cannot be masked and must be accepted by the 68000.**
- **When an interrupt is accepted $I_2\ I_1\ I_0$ (Interrupt mask) is set to the current interrupt level.**

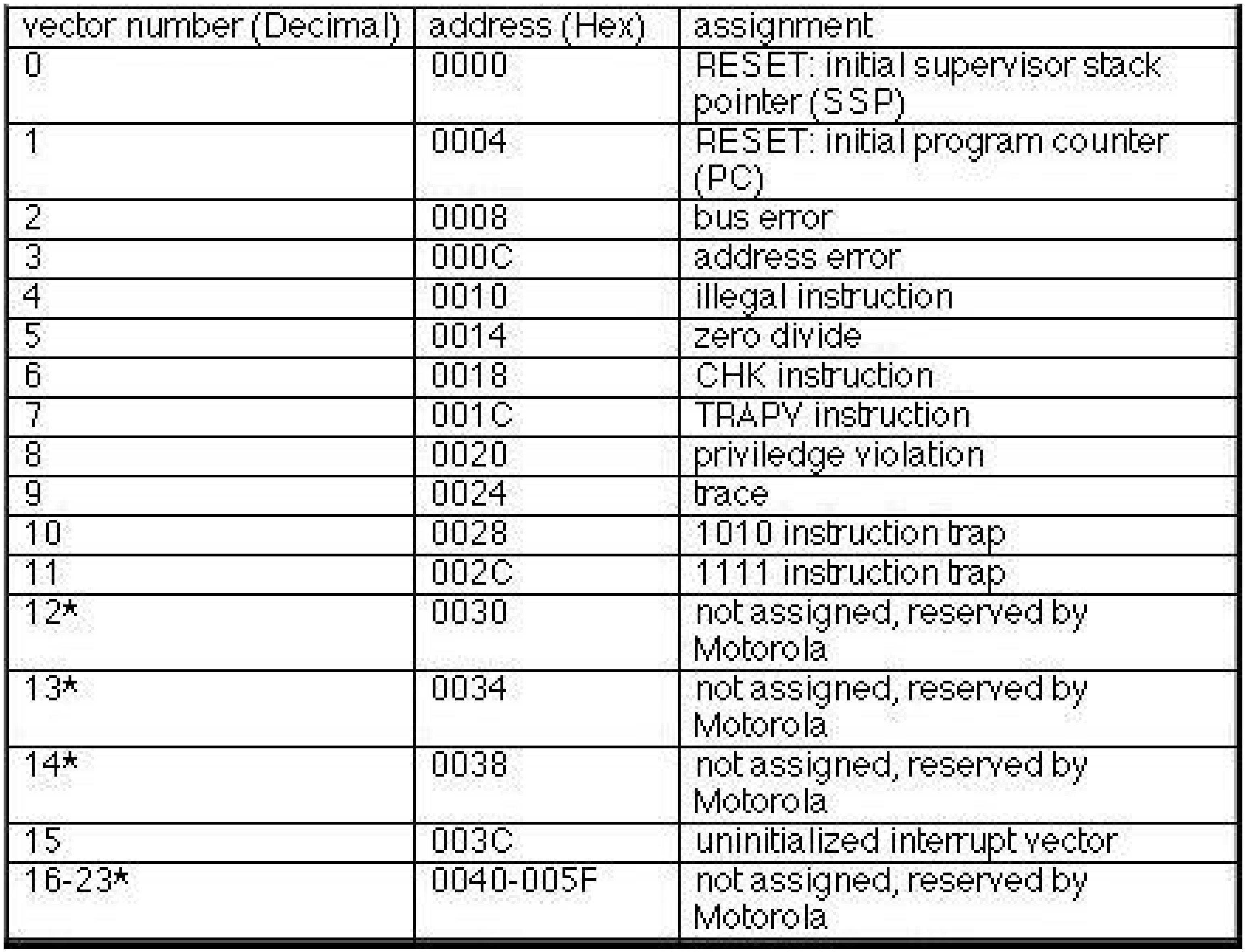

system byte
user byte (CCR)
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
T S I2 I1 I0 X N Z V C
Interrupt level
Supervisor mode
Allows a privileged mode of execution which is essential for a multi-user environment.
Trace
Sets a post-instruction routine into action.

Status Register: The System Part



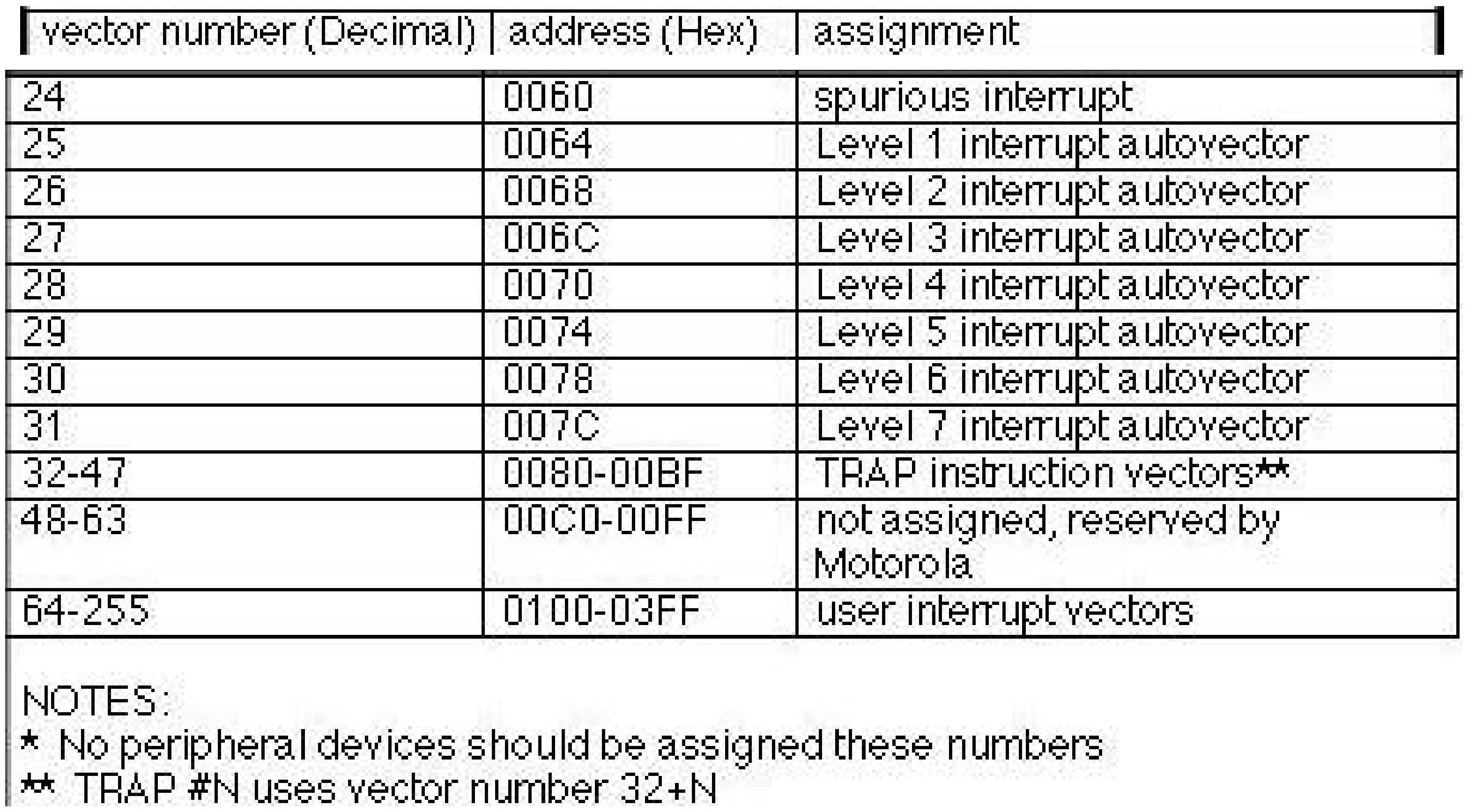
bits	function
11,12,14	not used
8,9,10	interrupt mask a priority scheme to determine who has control of the computer
13	supervisor set to 0 if user, set to 1 if supervisor
15	trace set to 1 if program is to be single stepped

68000 Exception Vector Table



vector number (Decimal)	address (Hex)	assignment
0	0000	RESET: initial supervisor stack pointer (SSP)
1	0004	RESET: initial program counter (PC)
2	0008	bus error
3	000C	address error
4	0010	illegal instruction
5	0014	zero divide
6	0018	CHK instruction
7	001C	TRAPV instruction
8	0020	priviledge violation
9	0024	trace
10	0028	1010 instruction trap
11	002C	1111 instruction trap
12*	0030	not assigned, reserved by Motorola
13*	0034	not assigned, reserved by Motorola
14*	0038	not assigned, reserved by Motorola
15	003C	uninitialized interrupt vector
16-23*	0040-005F	not assigned, reserved by Motorola

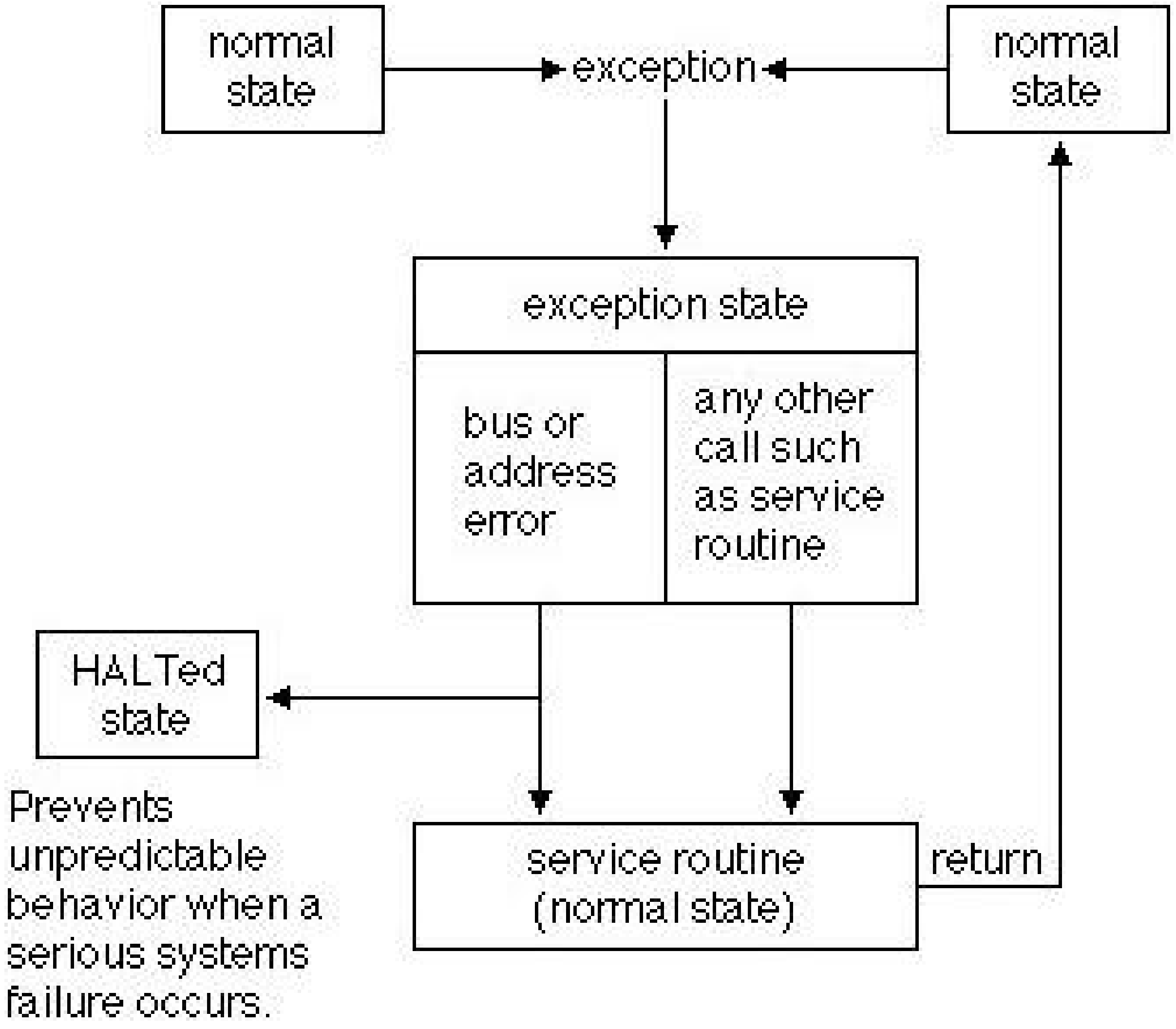
68000 Exception Vector Table (continued)



vector number (Decimal)	address (Hex)	assignment
24	0060	spurious interrupt
25	0064	Level 1 interrupt autovector
26	0068	Level 2 interrupt autovector
27	006C	Level 3 interrupt autovector
28	0070	Level 4 interrupt autovector
29	0074	Level 5 interrupt autovector
30	0078	Level 6 interrupt autovector
31	007C	Level 7 interrupt autovector
32-47	0080-00BF	TRAP instruction vectors**
48-63	00C0-00FF	not assigned, reserved by Motorola
64-255	0100-03FF	user interrupt vectors

NOTES:
* No peripheral devices should be assigned these numbers
** TRAP #N uses vector number 32+N

68000 Execution States


normal state
exception
normal state
exception state
bus or address error
any other call such as service routine
HALTed state
Prevents unpredictable behavior when a serious systems failure occurs.
service routine (normal state)
return

Interrupt Handling Steps

When an interrupt is requested by I/O and accepted by the CPU…

1. CPU finishes executing the current instruction
2. Acknowledge the acceptance of the interrupt to the I/O device.
3. Device provides interrupt vector after the acknowledgement.
4. Determine the start address of ISR (which interrupt vector).
   - $\Rightarrow$ Usually from the exception vector table in memory.
5. Push PC and Status Register, SR on stack.
6. Initialize the status register (for the exception routine)
   - $\Rightarrow$ Usually, set S = 1, T = 0, update interrupt level in SR for external exceptions to the current accepted interrupt level
7. Load ISR start address into PC
8. ISR proceeds to execute like a normal subroutine except it must end with the instruction:

   RTE ReTurn from Exception

   (similar to RTS, pops PC and SR from system stack)

RTE   Return from Exception (M68000 Family)   RTE

Operation: If Supervisor State
  Then $(SP) \rightarrow SR$; $SP + 2 \rightarrow SP$; $(SP) \rightarrow PC$; $SP + 4 \rightarrow SP$; Restore State and Deallocate Stack According to (SP)
Else TRAP

Assembler Syntax: RTE

Attributes: Unsized

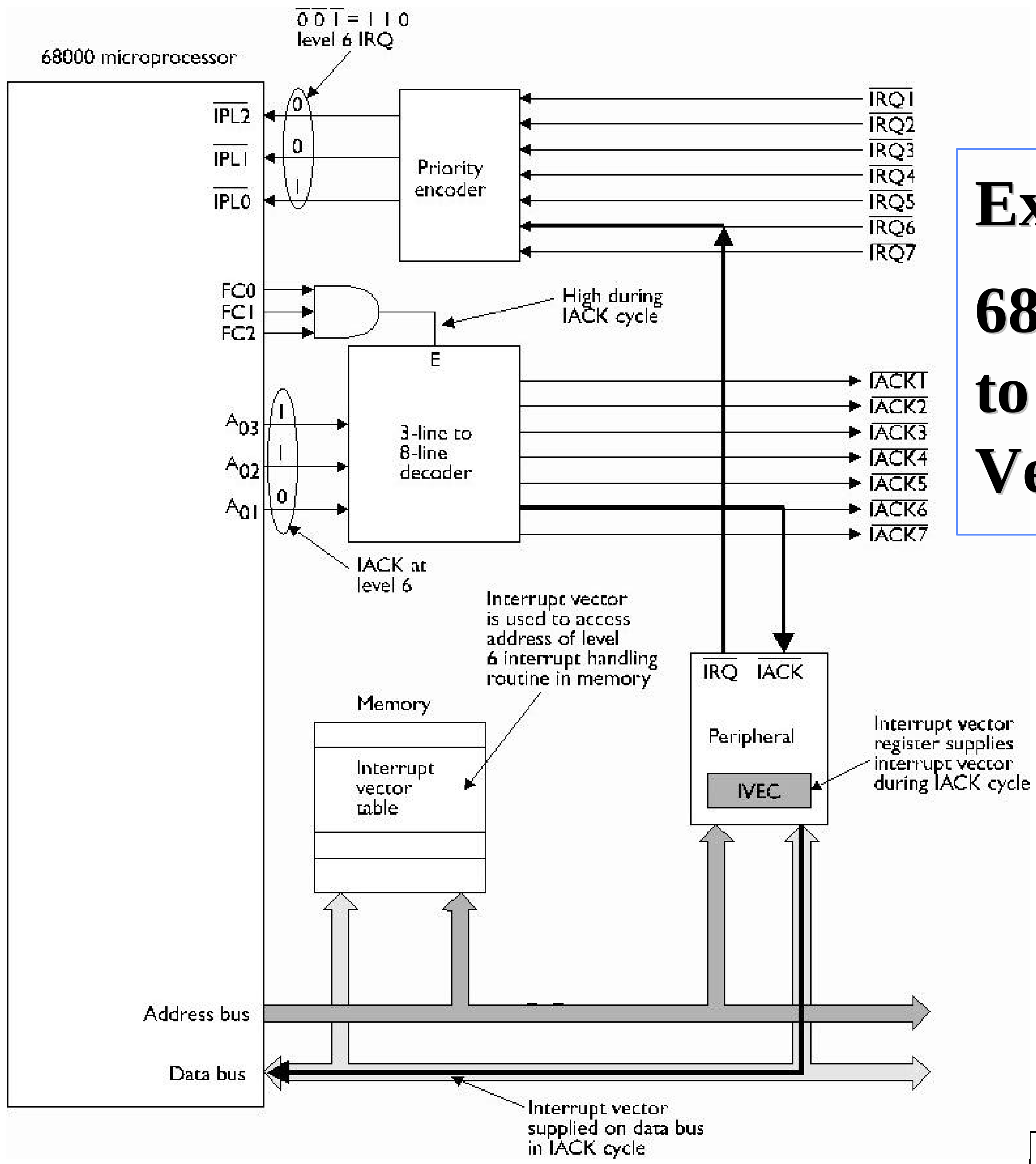
Description: Loads the processor state information stored in the exception stack frame located at the top of the stack into the processor. The instruction examines the stack format field in the format/offset word to determine how much information must be restored.

Condition Codes:

Set according to the condition code bits in the status register value restored from the stack.

Example:

68000 Responding to a level 6 Vectored Interrupt



EECC250 - Shaaban

Format of 68230 Port General Control Register, PGCR

PGCR7	PGCR6	PGCR5	PGCR4	PGCR3	PGCR2	PGCR1	PGCR0
Port Mode Control		H34 Enable	H12 Enable	H4 sense	H3 sense	H2 sense	H1 sense
00 Mode 0 01 Mode 1 10 Mode 2 11 Mode 3		0 Disable 1 Enable		0 Active low 1 Active high			

Example:

PGCR = %00010000

Means:

Mode 0, Unidirectional 8-bit, separate PA & PB

H34 handshaking disabled

H12 handshaking enabled

H4-H4 active low

68230 Port Status Register, PSR

- Reflects activity of the handshake lines

PSR7	PSR6	PSR5	PSR4	PSR3	PSR2	PSR1	PSR0
H4 Level	H3 Level	H2 Level	H1 Level	H4S	H3S	H2S	H1S

← Follow level of line → ← Set by line depending on mode →

PSR0-PSR3 must be cleared by the program by writing a 1 onto them

PACR Format of Port A Control Register in Mode 0

PACR7 PACR6	PACR5 PACR4 PACR3	PACR2	PACR1 PACR0
Submode: 00 submode 0 01 submode 1 10 submode 1x	H2 Control 0xx Edge-sensitive input 100 output- negated 101 output - asserted 110 output - interlocked handshake 111 Output - pulsed handshake	H2 Interrupt 0 Disabled 1 Enabled	H1 Control 0X H1 interrupt disabled 10 H1 interrupt enabled XX

Example: PACR = %00000010
PADDR = %00000000

Means: Port A is used as an input port
Submode 0 (Double Buffered input)
H2 Edge-sensitive
H2 interrupt disabled
H1 interrupt enabled

68230 Port Service Request Register PSRR (PIT + $02)

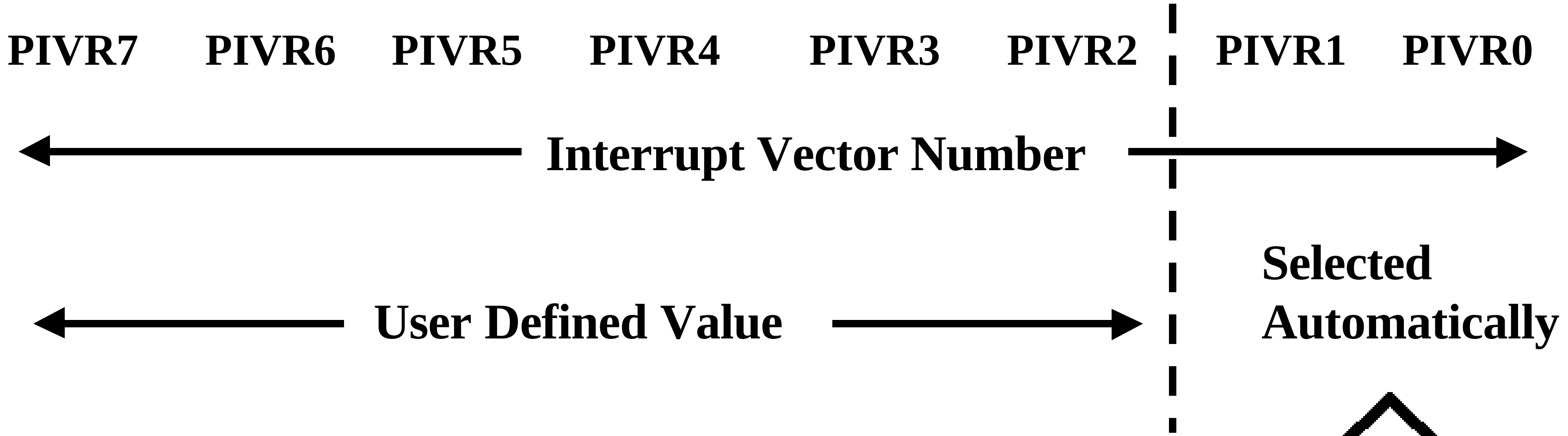
- Determines PIT interrupt/DMA settings

PSRR7	PSRR6	PSRR5	PSRR4	PSRR3	PSRR2	PSRR1	PSRR0
X	DMA Control		Interrupt Control		Port Priority Control		

PSRR4	PSRR3	
0	0	No interrupt support
0	1	Autovectored interrupts
1	0	
1	1	Vectored interrupts supported

Port Interrupt Priority			Order of Priority			
PSRR2	PSSR1	PSSR0	Highest			Lowest
0	0	0	H1S	H2S	H3S	H4S
0	0	1	H2S	H1S	H3S	H4S
0	1	0	H1S	H2S	H4S	H3S
0	1	1	H2S	H1S	H4S	H3S
1	0	0	H3S	H4S	H1S	H2S
1	0	1	H3S	H4S	H2S	H1S
1	1	0	H4S	H3S	H1S	H2S
1	1	1	H4S	H3S	H2S	H1S

Format of 68230 Port Interrupt Vector Register PIVR (PIT+$0A)



Interrupt Source	PIVR1	PIVR0
H1	0	0
H2	0	1
H3	1	0
H4	1	1

```
PIT      EQU    $0FF000     Base Address of PI/T
PGCR     EQU    PIT         Port General Control Register
PSRR     EQU    PIT+2       Port service request register
PADDR    EQU    PIT+4       Data direction register A
PBDDR    EQU    PIT+6       Data direction register B
PCDDR    EQU    PIT+$08     Data direction register C
PIVR     EQU    PIT+$0A     Port Interrupt Vector register
PACR     EQU    PIT+$0C     Port A control register
PBCR     EQU    PIT+$0E     Port B control register
PADR     EQU    PIT+$10     Port A data register
PBDR     EQU    PIT+$12     Port B data register
PCDR     EQU    PIT+$18     Port C data register
PSR      EQU    PIT+$1A     Port status register
TCR      EQU    PIT+$20     Timer control register
TIVR     EQU    PIT+$22     Timer interrupt vector register
CPR      EQU    PIT+$24     Dummy address of preload register
CPRH     EQU    PIT+$26     Timer preload register high
CPRM     EQU    PIT+$28     Timer preload register middle
CPRL     EQU    PIT+$2A     Timer preload register low
CNTR     EQU    PIT+$2C     Dummy address of timer register
CNTRH    EQU    PIT+$2E     Timer register high
CNTRM    EQU    PIT+$30     Timer register middle
CNTRL    EQU    PIT+$32     Timer register low
TSR      EQU    PIT+$34     Timer status register
```

Addresses of Port Related Registers

Addresses of Timer Related Registers

68230 Registers Address Equates

Interrupt-Driven PIT Input Example

- Input one byte from port A and buffers it in memory, whenever an interrupt is received on line H1

```
* Main Program
H1_VEC    EQU        68                 PIT Exception  vector number
H1_V_A    EQU        H1_VEC*4           ISR Exception vector table address
PGCRM     EQU        %00010000          Mode 0,  submode 00
PACRM     EQU        %00000010          PGCR value to enable H1 interrupt
          ORG        $1000
MAIN      MOVEA.L    #$07FFE,A7         Initialize SP
          LEA        H1_ISR,A0          Load A0 with address of PIT Routine
          MOVE.L     A0,H1_V_A          Put ISR address in vector table
          MOVE.B     #H1_VEC,PIVR       Initialize    PIVR  with interrupt vector
          MOVE.B     #PGCRM,PGCR        Initialize PGCR
          MOVE.B     #PACRM,PACR        Initialize port A
          MOVE.B     #$00, PADDR        Set Port A as input
          MOVE.B     #%00011000,PSRR    Enable vectored interrupts in PSRR
          LEA        BUFFER,A1          Load DATA address in A1
          ANDI       #$0F0FF,SR         Enable interrupts in SR
          LOOP       NOP                Fake loop just for example
          BRA        LOOP               main program doing other tasks here
```

Interrupt-Driven PIT Input Example

```
****    PIT Interrupt Service Routine  H1_ISR
*
H1_ISR   ORI          #$0700,SR     Disable interrupts
         MOVE.B       PADR,D1       Get a byte from port A
         MOVE.B       D1,(A1)+      Store byte in memory buffer
         ANDI         #$0F0FF,SR    Enable interrupts
         RTE                        Return from exception
         STOP         #$2700
         ORG          $2000
BUFFER   DS.B         1000          reserve 1000 bytes for buffer
         END
```

Timer Control Register (TCR) Value To Enable Periodic Timer Interrupt

TCR7	TCR6	TCR5	TCR4	TCR3	TCR2	TCR1	TCR0
Tout/TIACK* Control			Zero-detect Control	None	Clock Control		Timer Enable
1	0	1	0	0	0	0	1
PC3/Tout used as timer interrupt request line PC7/TIACK* used to acknowledge timer interrupts			After ZD, counter restarts from initial preloaded value		PC2/Tin not used counter clock CLK/32		Enable timer

Periodic Timer Interrupts Example

- **The subroutine T_SET preloads the timer with an initial value, and enables timer interrupt.**
- **Once the timer is enabled by calling T_SET, an interrupt is generated periodically to perform the tasks in the timer interrupt service routine, T_ISR.**

```
*
* Timer setup subroutine:
T_VEC    EQU        70              Timer Exception  vector number
T_V_A    EQU        T_VEC*4         ISR Exception vector table address
         ORG        $1000
T_SET    LEA        T_ISR,A0        Load A0 with address of Timer ISR
         MOVE.L     A0,T_V_A        Put Timer ISR address in vector table
         MOVE.B     #T_VEC,TIVR     Initialize  TIVR  with  interrupt vector
         MOVE.L     #$00FFFFFF,D0   Set maximum count
         MOVE.L     D0,CPR          preload count value in CPR
         MOVE.B     #%10100001      set up TCR, enable timer
         RTS
* Timer interrupt service routine.
T_ISR    MOVE.B     #1,TSR          Clear ZDS bit in TSR
*        .  .  .                      .  .  .
*        .  .  .                    Do tasks needed in ISR
         RTE
```