

F²MC-16LX Standby Cancel Failure

1	Summary	2
2	List of affected products.....	2
3	Description	3
4	Countermeasure	3
5	Structure of programs not affected	4
6	Appendix	7
6.1	Operation of F2MC-16LX CPU.....	8
6.2	Timing of Standby-Cancel Failure and operation after this failure	10
6.3	Timing of software workaround.....	12
6.4	Software workaround in C programming language.....	18
6.5	Examples of programs not affected by Standby-Cancel problem.....	20
6.6	Examples of programs affected by Standby-Cancel problem	22
6.7	Flowchart to decide whether Standby-Cancel problem affects existing software	23

1. Summary

The definition of Standby Cancel Failure is that the CPU will execute wrong instructions when an interrupt is executed during transition to Standby mode *0 at a certain time. Fujitsu can reproduce this phenomenon Fujitsu internally and has found the cause.

*0:Definition of Standby mode

Main sleep mode, PLL sleep mode, Sub-sleep mode

Time base timer mode, Watch mode, Main watch mode

Main stop mode, PLL stop mode, Sub-stop mode

*Main watch mode is only for MB90370 series.

In the following cases, no problem occurs:

- Standby mode is not used
- Standby mode is released only by external reset

2. List of affected products

Table1: List of affected products

Series name	Products name
MB90330series	MB90V330,MB90F334,MB90F337
MB90340series	MB90V340,MB90V340S,MB90F347D,MB90F347DS
MB90370series	MB90V370,MB90F372,MB90372
MB90385series	MB90F387,MB90F387S,MB90387,MB90387S
MB90390series	MB90V390,MB90V390H,MB90F394H
MB90405series	MB90MV405,MB90MF408,MB90M408,MB90M407
MB90420/425series	MB90V420G,MB90F423GA,MB90F423GB,MB90F423GC,MB90F428GA MB90F428GB,MB90F428GC,MB90423GA,MB90427GA,MB90428GA MB90423GB,MB90427GB,MB90428GB,MB90423GC,MB90427GC,MB90428GC
MB90440series	MB90V440G,MB90F443G
MB90455series	MB90F457,MB90F456,MB90F455,MB90F457S,MB90F456S,MB90F455S MB90457,MB90456,MB90455,MB90457S,MB90456S,MB90455S
MB90470series	MB90V470B,MB90F474L,MB90F474H,MB90F476A,MB90478,MB90477 MB90474,MB90473
MB90480series	MB90V480,MB90F481,MB90F482
MB90495series	MB90V495G,MB90F497G,MB90F498G,MB90497G
MB90800series	MB90V800,MB90F804,MB90803
MB90810series	MB90V810,MB90F818,MB90F818C,MB90F812,MB90F812C
MB90850series	MB90857GA,MB90857GB,MB90857GC
MB90865series	MB90F867,MB90F867S

3. Description

Standby Cancel failure will occur during a transition from CPU run mode *1 to Standby mode. Standby transition is initiated by setting a LPMCR register. If an interrupt is occurs during transition to Standby mode, this transition is stopped and the CPU returns to Run mode. But if this interrupt is occurs at a certain time, Standby Cancel failure may show up during returning to CPU run mode.

The problem occurs only, when the interrupt arrives at a certain time *2 right after the execution of the instruction causing the transition to Standby mode. If the interrupt occurs during this time, an inconsistency will be caused between the Queue Counter of Bus Control Unit and the Program Counter (PC) of the CPU. The Queue counter in the Bus Control Unit is one byte address ahead of the PC of the CPU. Hence, the MCU may execute wrong code when it returns to CPU run mode.

This Standby Cancel Failure can be solved with a software work around as described in chapter 4. (“Countermeasure”). A Reset *3 can also solve this failure.

About details of the Standby Cancel Failure, please refer to Appendix 6.1-6.3.

*1: CPU Run mode includes follows modes.

Main Clock Run mode, PLL Clock Run mode, Sub Clock Run mode

*2: The definition of certain duration is as follows.

Single Chip mode: 3 machine cycles

External Bus mode: 4 machine cycles

*3: Reset is defined as follows.

Power On reset, Watchdog reset, External reset, Software reset

4. Countermeasure

The Standby Cancel Failure can be solved with the following software workaround for products on Table 1. (“List of affected products”).

The software workaround can solve this problem completely.

Countermeasure: Add following instructions within the dotted line frame right after the Standby mode instruction.

<pre> MOV LPMCR,#xxh ;Standby mode transition instruction NOP NOP JMP LABEL LABEL: MOV A,#010H ;Customer application program </pre>

Today there is no plan to fix this failure for devices in Table 1. (“List of affected products”). Future products will be fixed. Please use this software workaround.

We have a plan to prepare a check tool as follows.

-Standby Cancel Failure check tool will be prepared by Begin of January in 2003.

-Installation of this check tool to Softune V3 will be by Begin of March in 2003.

*The Standby-Cancel Failure check tool will have restrictions. Information about the restrictions will be available at the release date.

5. Structure of programs not affected

If customer's software has the following instruction structure, it can be solved without the countermeasure software workaround. Please refer to Appendix.6.5 for more program examples.

(1) Interrupt for Standby release does not happen during the critical time period *2 right after execution of Standby mode transition instruction.

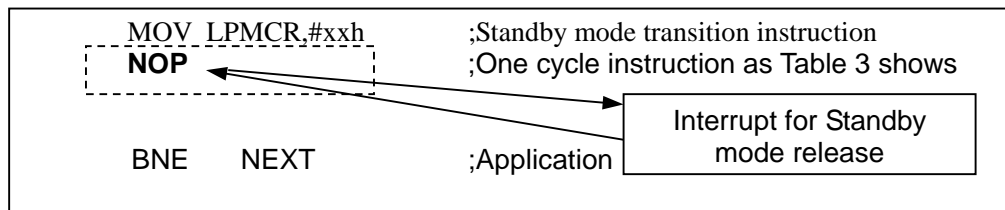
(2) There is an instruction as listed in Table 2 immediately after the instruction causing the Standby transition. These instructions do not update the Queue Counter.

MOV LPMCR,#xxh	;Standby mode transition instruction
CLRB EIRR:0	;Instruction which does not update the
	; Queue counter (please see Table 2).
MOV A,#010H	;Application

Reason: Instructions listed in Table 2 do not update the Queue Counter during releasing Standby mode. Therefore, the Standby-Cancel Failure is avoided.

Please see the software workaround 3 for more detail.

(3) There is an instruction as listed in Table 3 right after the Standby mode transition instruction and an Interrupt routine is executed by the Interrupt.



Reason: Since the Instruction Request is masked by the interrupt execution the Standby-Cancel failure does not happen.

Please see the software workaround 2 for more detail.

Table 2: Table of Instructions that do not update the Queue Counter

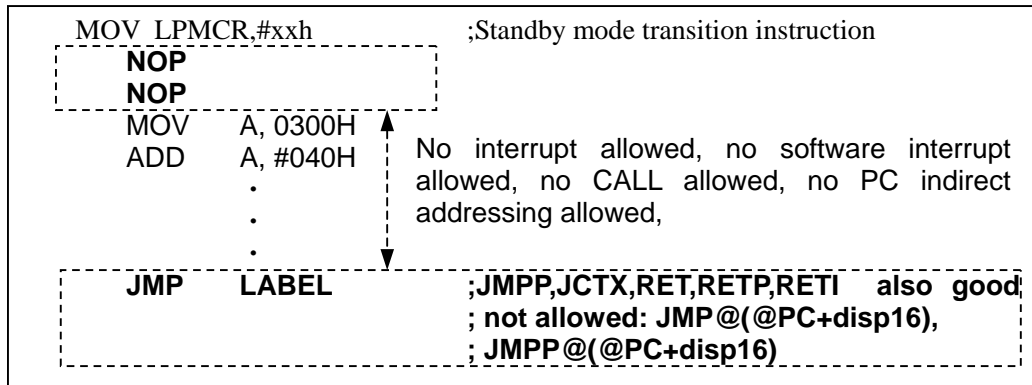
ADDC A	XCHW A,@RWj	CMPW A,@RWj	RORC,A
ADDDC A	XCHW A,@RWj+	CMPW A,@RWj+	RORC @RWj
SUBC A	XCHW RWi,@RWj	CMPL A,@RWj	RORC @RWj+
SUBDC A	XCHW RWi,@RWj+	CMPL A,@RWj+	ROLC,A
ADDW A	MOVL A,@RWj	MULU A	ROLC @RWj
SUBW A	MOVL A,@RWj+	MULU A,@RWj	ROLC @RWj+
DIVU A	MOVL @RWj,A	MULU A,@RWj+	JMP @(@RWj)
MULU A	MOVL @RWj+,A	MULUW A	JMP @(@RWj+)
MULUW A	ADD A,@RWj	MULUW A,@RWj	JMPP @(@RWj)
NOT A	ADD A,@RWj+	MULUW A,@RWj+	JMPP @(@RWj+)
ANDW A	ADD @RWj,A	DIVU A,@RWj	CALL @(@RWj)
ORW A	ADD @RWj+,A	DIVU A,@RWj+	CALL @(@RWj+)
XORW A	ADDC A,@RWj	DIVUW A,@RWj	CALLP @(@RWj)
NOTW A	ADDC A,@RWj+	DIVUW A,@RWj+	CALLP @(@RWj+)
NEG A	ADDC A,ear	AND A,@RWj	CALLP @ear
NEGW A	SUB A,@RWj	AND A,@RWj+	DBNZ @RWj,REL
ASRW A	SUB A,@RWj+	AND @RWj,A	DBNZ @RWj+,REL
LSRW A / SHRW A	SUB @RWj,A	AND @RWj+,A	DWBZ @RWj,REL
LSLW A / SHLW A	SUB @RWj+,A	OR A,@RWj	DWBZ @RWj+,REL
SWAP	SUBC A,@RWj	OR A,@RWj+	INT9
SWAPW / XCHW A,T	SUBC A,@RWj+	OR @RWj,A	PUSHW A
EXTW	SUBC A,ear	OR @RWj+,A	PUSHW AH
MOV A,Ri	ADDW A,@RWj	XOR A,@RWj	PUSHW PS
MOV A,ear	ADDW A,@RWj+	XOR A,@RWj+	AND CCR,#imm8
MOV A,@RWj	ADDW @RWj,A	XOR @RWj,A	OR CCR,#imm8
MOV A,@RWj+	ADDW @RWj+,A	XOR @RWj+,A	MOVEA RWi,@RWj
MOVX A,Ri	ADDCW A,@RWj	NOT @RWj	MOVEA RWi,@RWj+
MOVX A,ear	ADDCW A,@RWj+	NOT @RWj+	MOVEA A,@RWj
MOVX A,@RWj	ADDCW A,ear	ANDW A,@RWj	MOVEA A,@RWj+
MOVX A,@RWj+	SUBW A,@RWj	ANDW A,@RWj+	MOV A,DTB
MOV @RWj,A	SUBW A,@RWj+	ANDW @RWj,A	MOV A,ADB
MOV @RWj+,A	SUBW @RWj,A	ANDW @RWj+,A	MOV A,SSB
MOV Ri,@RWj	SUBW @RWj+,A	ORW A,@RWj	MOV A,USB
MOV Ri,@RWj+	SUBCW A,@RWj	ORW A,@RWj+	SETB io:bp
MOV @RWj,Ri	SUBCW A,@RWj+	ORW @RWj,A	CLRB io:bp
MOV @RWj+,Ri	SUBCW A,ear	ORW @RWj+,A	CBNE @RWj,#imm8,rel
MOV @RWj,#imm8	ADDL A,@RWj	XORW A,@RWj	CBNE @RWj+,#imm8,rel
MOV @RWj+,#imm8	ADDL A,@RWj+	XORW A,@RWj+	CWBNE @RWj,#imm16,rel
XCH A,@RWj	SUBL A,@RWj	XORW @RWj,A	CWBNE @RWj+,#imm16,rel
XCH A,@RWj+	SUBL A,@RWj+	XORW @RWj+,A	MUL A,ear
XCH Ri,@RWj	INC @RWj	NOTW @RWj	MUL A,@RWj
XCH Ri,@RWj+	INC @RWj+	NOTW @RWj+	MUL A,@RWj+
MOVW A,RWi	DEC @RWj	ANDL A,@RWj	MULW A,RWi
MOVW A,ear	DEC @RWj+	ANDL A,@RWj+	MULW A,@RWj
MOVW A,@RWj	INCW @RWj	ORL A,@RWj	MULW A,@RWj+
MOVW A,@RWj+	INCW @RWj+	ORL A,@RWj+	DIV A,@RWj
MOVW @RWj,A	DECW @RWj	XORL A,@RWj	DIV A,@RWj+
MOVW @RWj+,A	DECW @RWj+	XORL A,@RWj+	DIVW A,@RWj
MOVW RWi,@RWj	INCL @RWj	NEG @RWj	DIVW A,@RWj+
MOVW RWi,@RWj+	INCL @RWj+	NEG @RWj+	MUL A
MOVW @RWj,RWi	DECL @RWj	NEGW A	MULW A
MOVW @RWj+,RWi	DECL @RWj+	NEGW @RWj	DIV A
MOVW @RWj,#imm16	CMP A,@RWj	NEGW @RWj+	
MOVW @RWj+,#imm16	CMP A,@RWj+	NRML A,R0	

Table 3: Table of One-Cycle instructions

MOVN A,#imm4	CMPW A	ZEXTW	MOV DTB,A
MOVW A,SP	NOP	MOV A,PCB	MOV ADB,A
MOVW SP,A	EXT	MOV A,DPR	MOV SSB,A
CMP A	ZEXT	MOV DPR,A	MOV USB,A

(4) Next instruction right after Standby mode transition instruction is executed by interrupt of Standby mode release.

Condition: There are more than two NOP right after Standby mode transition instruction and there is JMP*5, JMPP*5, JCTX, RET, RETP, RETI instruction before an instruction*4 that uses PC. However, interrupt is not executed before execution of JMP*5, JMPP*5, JCTX, RET, RETP, RETI instruction.



Reason: After Standby-Cancel Failure, the Queue Counter in the Bus Control Unit is one count ahead of PC of the CPU. Since the Queue Counter still provides instructions with correct order, the CPU runs correctly as long as the instruction*4 does not access the PC value. After the Queue Counter value is updated with the JMP instruction, the CPU can execute all instructions correctly.

*4: Instructions that PC is used

-Relative jump instruction

BZ/BEQ	BNT	CWBNE
BNZ/BNE	BLT	CBNE
BC/BLO	BGE	DBNZ
BNC/BHS	BLE	DWBNZ
BN	BGT	BBC
BP	BLS	BBS
BV	BHI	SBBS
BNV	BRA	
BT	CBNE	

-CALL, CALLP, CALLV instructions ,INT,INTP,INT9 instructions

-Instructions that PC relative is used as operand (@PC+disp16)

Example: MOV A,@PC+disp16

*5: JMP, JMPP instructions except

JMP @(@PC+disp16),JMPP @(@PC+disp16)

Please see Appendix 6.6 ("Examples of programs affected by Standby-Cancel problem") for more detailed examples of this failure and Appendix 6.7 ("Flowchart to decide whether Standby-Cancel problem affects existing software") to analyze existing programs.

6 Appendix

6.1	Operation of F2MC-16LX CPU.....	8
6.2	Timing of Standby-Cancel Failure and operation after this failure	10
6.3	Timing of software workaround.....	12
6.4	Software workaround in C programming language.....	18
6.5	Examples of programs not affected by Standby-Cancel problem.....	20
6.6	Examples of programs affected by Standby-Cancel problem	22
6.7	Flowchart to decide whether Standby-Cancel problem affects existing software	23

6 Appendix

6.1 Operation of F2MC-16LX CPU

Figure 1 shows a block diagram of the F²MC-16LX CPU and peripherals. The CPU is connected via the Bus Control Unit to the internal bus. The CPU requests instructions from the Bus Control Unit. The instructions are read from a 4 Byte Instruction Queue. A Queue Counter in the Bus Control Unit reads the instructions from the internal bus. The Bus Control Unit updates the Queue counter. Only in case of branch/jump instructions, the Queue Counter is updated with the value from the Program Counter in the CPU.

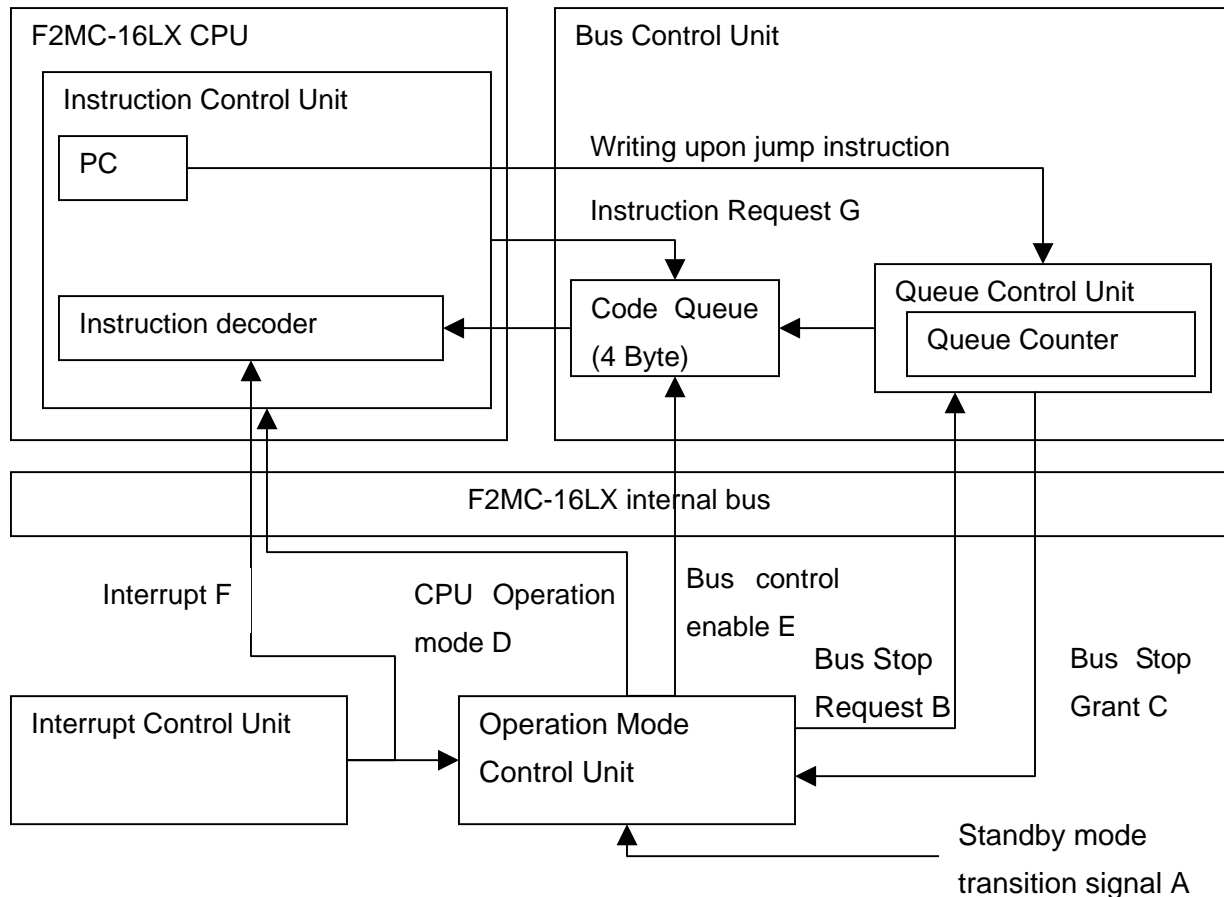


Figure 1: Block diagram of F2MC-16LX CPU and peripherals

Signals A to F control entering and leaving the Standby mode. The timing chart in Figure 2 and table 4 show the operation in which the Standby-Cancel failure does not occur, because the interrupt arrives after Bus Stop Grant signal C is high. The Standby Cancel failure will occur only if the interrupt arrives during one machine cycle that is right before of Bus Stop Grant signal C is high. The timing*6 of the Bus Stop Grant signal C becoming High is dependant on the state of instruction fetch. Figure 2 shows the example that Bus Stop Grant C becomes High after one machine cycle right after Bus Stop Request B is activated.

*6: Single chip mode:	1- 3 machine cycles after the activation of Bus stop request B
External Bus mode:	1- 4 machine cycles after the activation of Bus stop request B

Figure 2: Timing chart of entering/leaving Standby mode

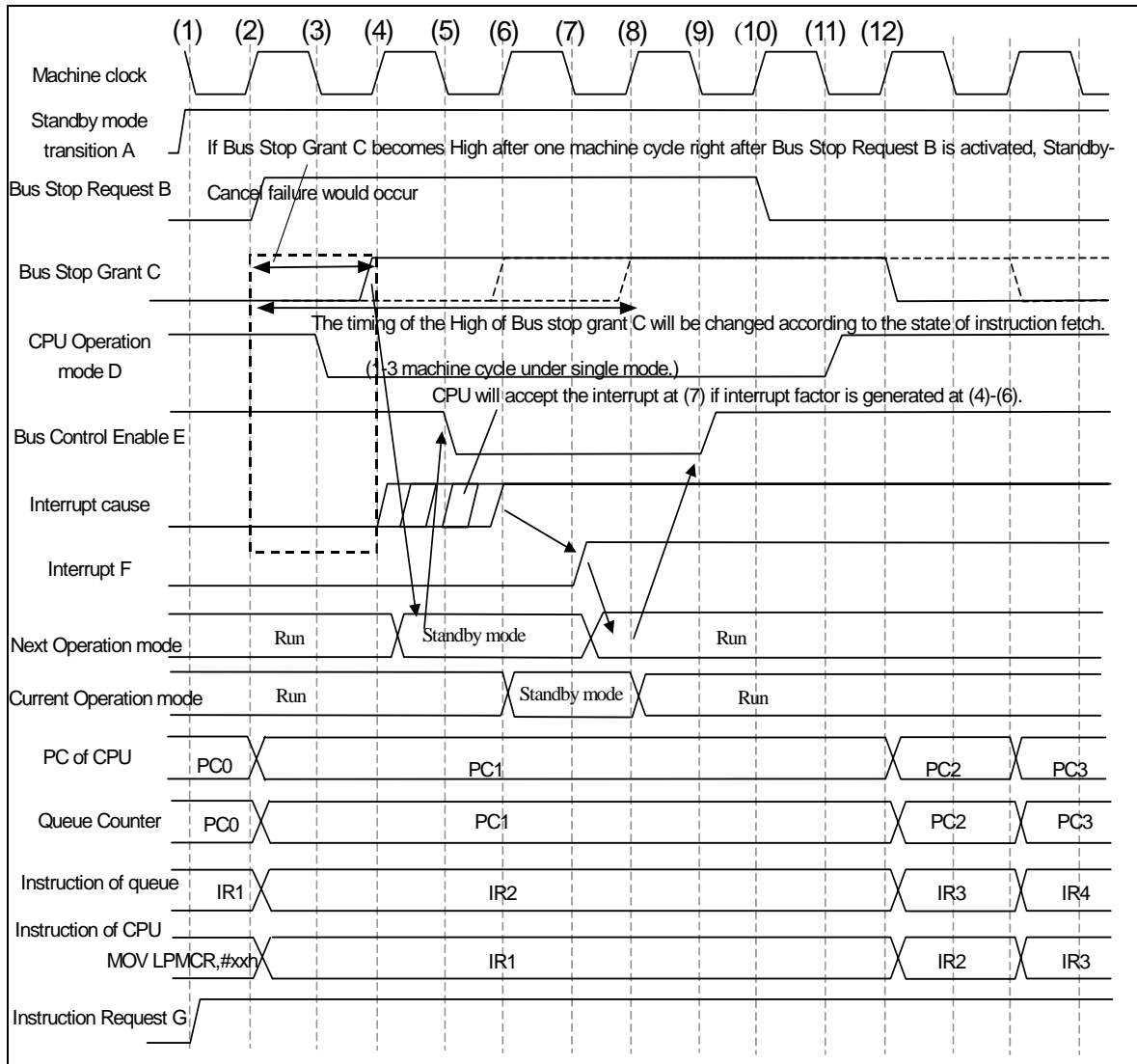


Table 4: Process of entering/leaving Standby mode

Timing	Operation	Interrupt
(1)	Standby mode transition A becomes High with the execution of Standby mode transition instruction (MOV LPMCR, #xxh) => Transition to Standby mode starts.	
(2)	Operation Mode Control Unit sets Bus Stop Request B to High => Bus Control Unit accepts Bus Stop Request	
(3)	Operation Mode Control Unit sets CPU Operation mode D to Low => CPU is stopped	
(4)	Bus Control Unit accepts Bus Stop Request and Bus Stop Grant C becomes High => Next Operation Mode becomes Standby mode	↑ Interrupt cause ↓
(5)	Operation Mode Control Unit set Bus Control Enable E to Low => Bus Control Unit is stopped	
(6)	Current Operation Mode becomes Standby mode	
(7)	Interrupt Control Unit sets Interrupt F to High because of interrupt at time (4)-(6) => Next Operation Mode becomes CPU RUN mode.	
(8)	Current Operation Mode decides becomes CPU run mode	
(9)	Operation Mode Control Unit sets Bus Control Enable E to High => Bus Control Unit restarts operation	
(10)	Operation Mode Control Unit sets Bus Stop Request B to Low => Bus Control Unit accepts cancellation of Bus Stop Request	
(11)	Operation Mode Control Unit sets CPU Operation Mode D to High => CPU operation restarts.	
(12)	Bus Control Unit starts Bus operation and Bus Stop Grant C becomes low => CPU starts execution of instruction	

6.2 Timing of Standby-Cancel Failure and operation after this failure

For the timing of the Standby-Cancel failure and operation after this failure, refer to Figure 3 (“Timing chart of entering/leaving Standby mode in case of failure”) and Table 5 (“Process of entering/leaving Standby mode”).

Figure 3: Timing chart of entering/leaving Standby mode in case of failure

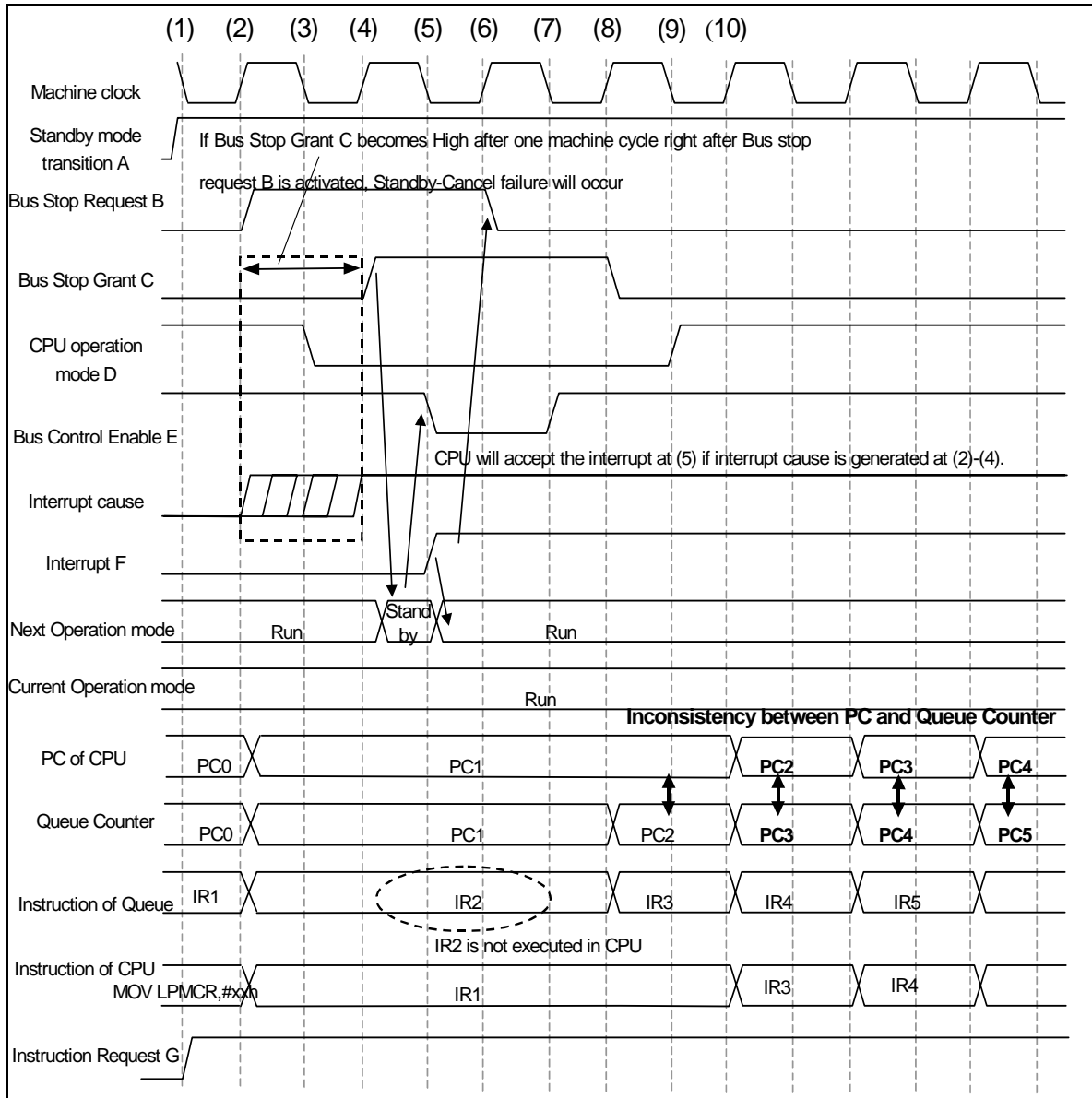


Table5: Process of entering/leaving Standby mode in case of failure

Timing	Operation	Interrupt
(1)	Standby mode transition A becomes High with the execution of Standby mode transition instruction (MOV LPMCR, #xxh) => Transition to Standby mode starts.	
(2)	Operation Mode Control sets Bus Stop Request B to High => Bus Control Unit accepts Bus Stop Request	Interrupt cause ↑ ↓
(3)	Operation Mode Control Unit sets CPU Operation mode D to Low => CPU is stopped	
(4)	Bus Control Unit accepts Bus Stop Request and Bus Stop Grant C becomes High => Next Operation mode becomes Standby mode	
(5)	Operation Mode Control sets Bus Control Enable E to Low => Bus Control Unit is stopped Interrupt F becomes High according to interrupt cause at time (2)-(4) => Next Operation mode becomes CPU RUN mode This is the actual cause of the Standby Cancel failure.	
(6)	Operation Mode Control Unit sets Bus Stop Request B to Low according to interrupt F at (5), because interrupt cancels Standby mode in Operation Mode Control Unit => Bus Control Unit accepts the cancellation of Bus Stop Request	
(7)	Operation Mode Control Unit sets Bus Control Enable E to High => Bus Control Unit restarts operation	
(8)	Bus Stop Grant C becomes Low because Bus Control Unit accepted the cancellation of Bus Stop Request => Queue Counter starts counting, though CPU is stopped, because Instruction Request G becomes High. As a result, Instruction of Queue changes to IR3 before execution of instruction IR2. A definition of Standby Cancel Failure is this phenomenon.	
(9)	Operation Mode Control Unit sets CPU Operation mode D to High => CPU restarts operation	
(10)	After this, PC of CPU starts, but Queue Counter in Bus Control Unit is one count ahead of the PC of CPU. Therefore the IR3 instruction is executed instead of the IR2 instruction. As a result, MCU does not run correctly until Queue Counter is updated with the value of the PC.	

6.3 Timing of software workaround

Please see software workaround (1)-(3) to avoid this failure.

Software workaround (1)

If the interrupt does not cause the execution of an interrupt service routine*7 and the instruction after the Standby mode transition instruction is executed, the Standby Cancel Failure can be solved with the software workaround as defined in the dotted line frame. Figure 4 shows the corresponding timing chart.

```

(PC0) MOV LPMCR,#xxh ;Standby mode transition instruction
(PC1) NOP           ;NOP 1
(PC2) NOP           ;NOP 2
(PC3) JMP LABEL
LABEL:
      MOV A,#010H   ;Application
    
```

*7: If interrupt is disabled (I=0) or interrupt level IL priority of Standby mode release interrupt is lower than interrupt level mask register ILM (ILM < IL), it can be solved.

Figure 4: Timing chart of entering/leaving Standby mode with software workaround (1)

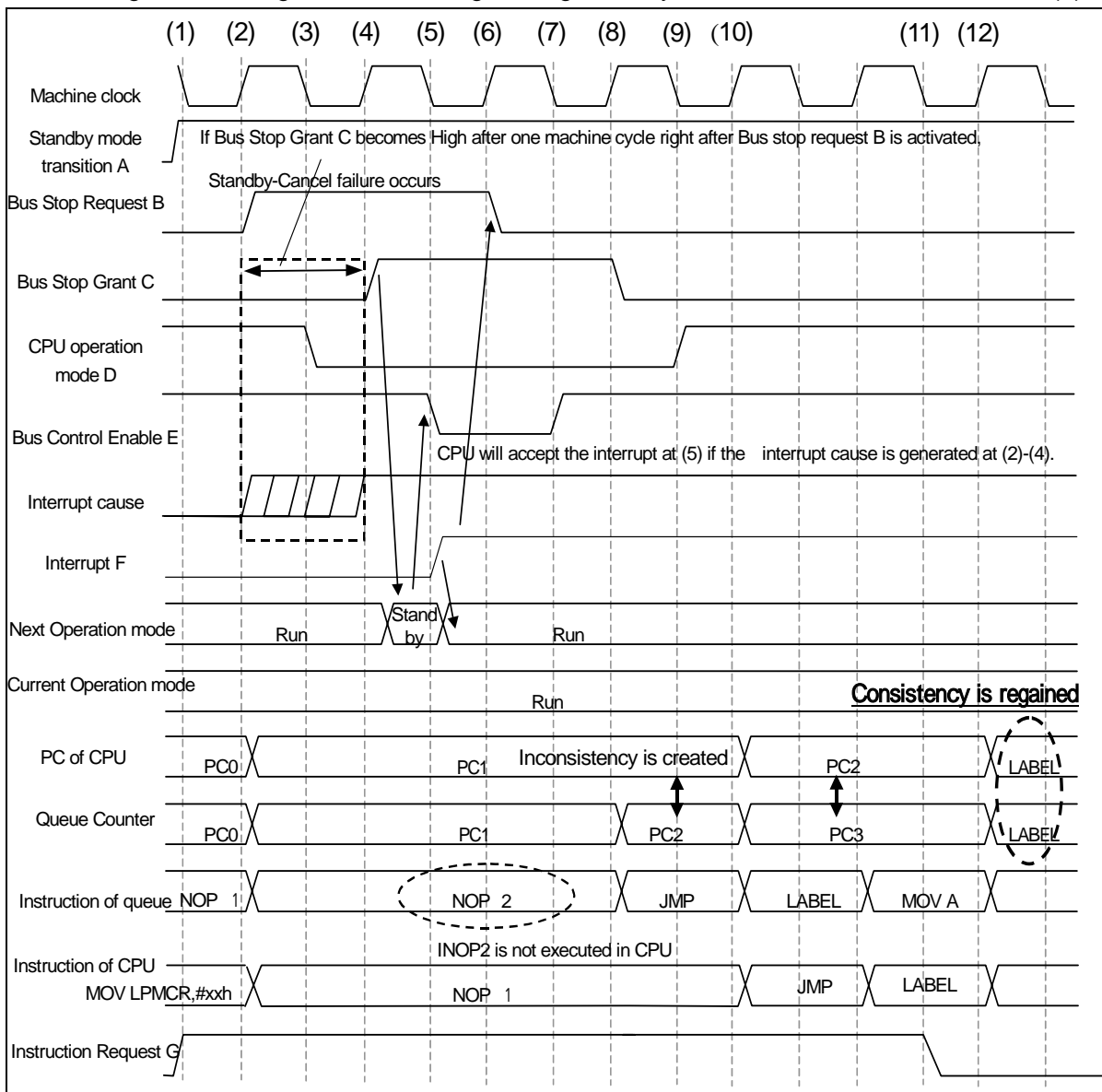


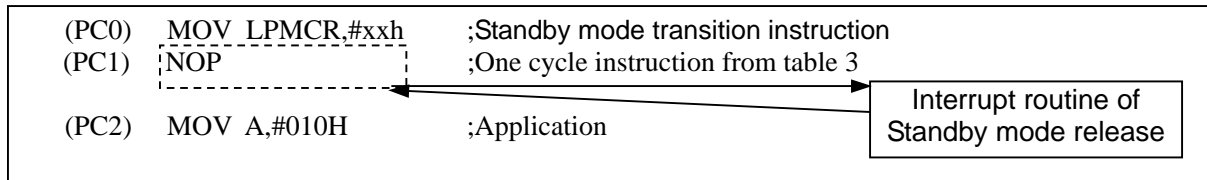
Table 6: Process of entering/leaving Standby mode with software workaround

Timing	Operation	Interrupt
(1)	Standby mode transition A becomes High with the execution of Standby mode transition instruction (MOV LPMCR, #xxh) => Transition to Standby mode starts.	
(2)	Operation Mode Control Units sets Bus Stop Request B to High => Bus Control Unit accepts Bus Stop Request	Interrupt cause ↑ ↓
(3)	Operation Mode Control Unit sets CPU Operation mode D to Low => CPU is stopped	
(4)	Bus Control Unit accepts Bus Stop Request and Bus Stop Grant C becomes High => Next operation mode becomes Standby mode	
(5)	Operation Mode Control Unit sets Bus Control Enable E to Low => Bus Control Unit is stopped Interrupt F becomes High according to interrupt cause at (2)-(4) => Next Operation mode becomes CPU RUN mode This is the actual cause of the Standby Cancel failure	
(6)	Operation Mode Control Unit sets Bus Stop Request B from to Low according to interrupt F at (5) because interrupt cancels Standby mode in Operation mode control => Bus Control Unit accepts the cancellation of Bus Stop Request	
(7)	Operation Mode Control Unit sets Bus Control Enable E to High => Bus Control Unit restarts operation	
(8)	Bus Stop Grant C becomes Low because Bus Control Unit accepted the cancellation of Bus Stop Request => Because Instruction Request G is High, the Queue Counter starts counting though the CPU is stopped. As a result, Instruction of Queue changes to IR3 before execution of the IR2 instruction. This is the Standby Cancel Failure phenomenon.	
(9)	Operation Mode Control Unit sets CPU Operation mode D to High => CPU restarts operation	
(10)	After this, PC of CPU starts but the Queue Counter in Bus Control Unit is one count ahead of the PC of the CPU. Therefore, the JMP instruction is executed instead of NOP2. NOP2 is not executed, but this is no problem.	
(11)	Instruction Request G becomes Low because this instruction is JMP	
(12)	Since by this jump instruction the Queue Counter value is updated with the PC value, the consistency between the Queue Counter and the PC of the CPU is regained. After this, the MCU runs correctly.	

Figure 4 and Table 6 shows the case that the interrupt routine is not executed when Standby mode is released by the interrupt. Software workaround 1 can solve this problem for each case that the interrupt routine is executed or not.

Software workaround (2)

If the interrupt causes the execution of an interrupt service routine *8 and the next instruction after the Standby mode transition instruction (enclosed by the dotted line frame) is an one-cycle instruction from table 3, the Standby Cancel Failure can be solved. Figure 5 shows this timing chart.



*8: If interrupt is enabled (I=1) and interrupt level IL priority of Standby mode release interrupt is higher than interrupt level mask register ILM (ILM < IL).

Figure 5: Timing chart of entering/leaving Standby mode with software workaround (2)

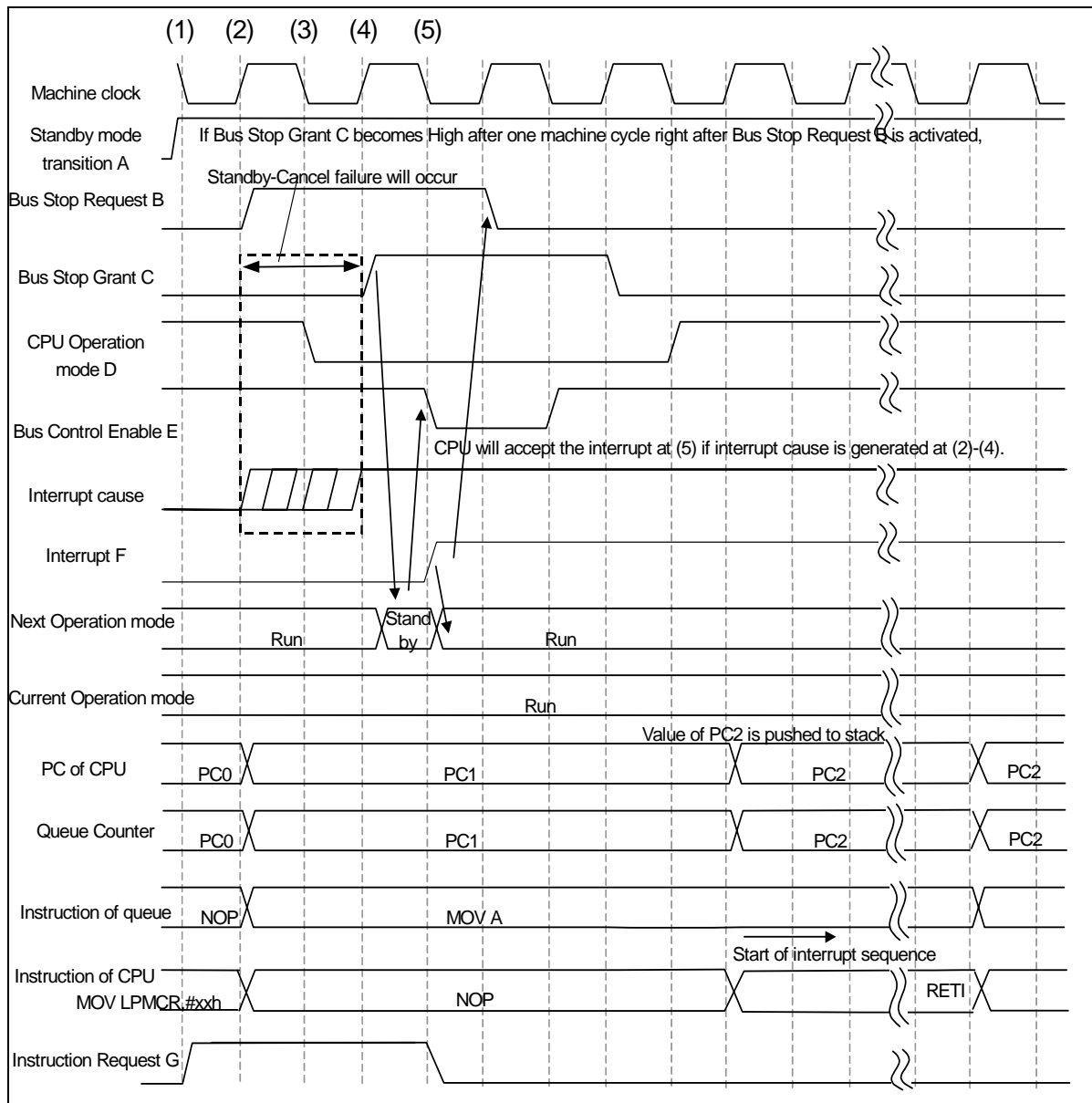


Table 7: Process of entering/leaving Standby mode with software workaround

Timing	Operation	Interrupt
(1)	Standby mode transition A becomes High with the execution of Standby mode transition instruction (MOV LPMCR, #xxh) => Transition to Standby mode starts.	
(2)	Operation Mode Control Unit sets Bus Stop Request B to High => Bus Control Unit accepts Bus Stop Request	Interrupt cause ↑ ↓
(3)	Operation Mode Control Unit sets CPU Operation mode D to Low => CPU is stopped	
(4)	Bus Control Unit accepts Bus Stop Request and Bus Stop Grant C becomes High => Next Operation mode becomes Standby mode	
(5)	Operation Mode Control Unit sets Bus Control Enable E to Low => Bus Control Unit is stopped Interrupt F becomes High according to interrupt cause at (2)-(4) => Next Operation mode becomes CPU RUN mode but Instruction Request G is masked by Interrupt F. In this case, since Instruction request G becomes Low, Standby-Cancel failure does not occur.	

Software workaround (3)

If there is an instruction that is defined in Table 2 right after Standby mode transition instruction, Standby-Cancel Failure does not occur as Figure 6 shows.

(PC0)	MOV LPMCR,#xxh	;Standby mode transition instruction
(PC1)	ADDC A	;Instruction which Queue counter is not updated, please see Table 2.
(PC2)	MOV A,#010H	;Application

Figure 6: Timing chart of entering/leaving Standby mode with software workaround (3)

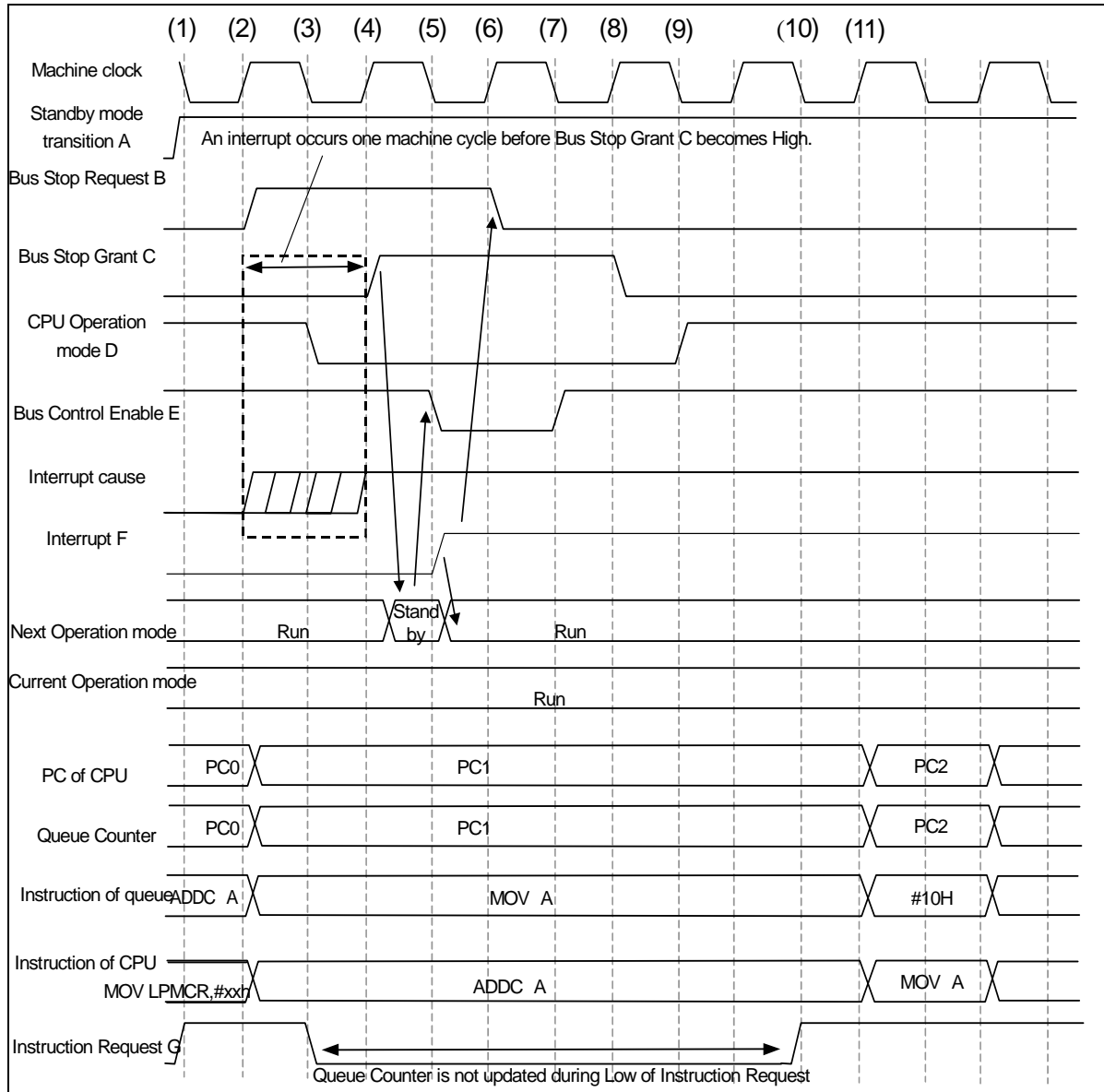


Table 8: Process of entering/leaving Standby mode with software workaround (3)

Timing	Operation	Interrupt
(1)	Standby mode transition A becomes High with the execution of Standby mode transition instruction (MOV LPMCR, #xxh) => Transition to Standby mode starts.	
(2)	Operation Mode Control Unit sets Bus Stop Request B to High => Bus Control Unit accepts Bus Stop Request	↑Interrupt cause ↓
(3)	Operation Mode Control Unit sets CPU operation mode D to Low => CPU is stopped Instruction Request G becomes Low because ADDC A instruction does not update the Queue Counter.	
(4)	Bus Control Unit accepts Bus Stop Request and Bus Stop Grant C becomes High => Next Operation mode becomes Standby mode	
(5)	Operation Mode Control Unit sets Bus Control Enable E to Low => Bus Control Unit is stopped Interrupt F becomes High according to Interrupt Cause at (2)-(4) => Next Operation mode becomes as CPU RUN mode	
(6)	Operation Mode Control Unit sets Bus Stop Request B to Low according to interrupt F at (5) because interrupt cancels Standby mode in Operation Mode Control Unit => Bus Control Unit accepts the cancellation of Bus Stop Request	
(7)	Operation Mode Control Unit sets Bus Control Enable E to High => Bus Control Unit restarts operation	
(8)	Bus Stop Grant C becomes Low because Bus Control Unit accepted the cancellation of Bus Stop Request => Since Instruction Request is low, the Queue Counter is not updated while the CPU is stopped. As a result, the Standby-Cancel failure does not happen.	
(9)	Operation Mode Control Unit sets CPU Operation mode D to High => CPU restarts operation	
(10)	Instruction Request G becomes High at the last cycle of ADDC A instruction	
(11)	The consistency between the Queue Counter and the Program Counter (PC) of the CPU is still kept after this.	

6.4 Software workaround in C programming language

If the software workaround will be done in the C programming language, please write it as follows.

The definition of LPMCR register and each bits are as follows.

```

/*Definition of LPMCR and each bits */
typedef union {
    unsigned char  byte;
    struct {
        unsigned char  STP:1;
        unsigned char  SLP:1;
        unsigned char  SPL:1;
        unsigned char  RST:1;
        unsigned char  TMD:1;
        unsigned char  CG:2;
        unsigned char  REV:1;
    } bit;
    } REG_LPMCR;

#pragma section IO=LPMCR,locate=0x0000a0
__io REG_LPMCR  lpmcr;

```

(1) Standby mode transition instruction should be made as a procedure and two built-in functions `__wait_nop()` should be added. However, it is necessary that no interrupts occur in the `enter_watch()` procedure.

```

void enter_watch() {
    lpmcr.byte = 0x10;      /* "0" is set at TMD bit of LPMCR */
    __wait_nop();          /*NOP is generated with a built-in function of _wait_nop() */
    __wait_nop();          /* 2nd NOP is generated */
}                          /* A function is closed */

```

The resulting assembler code of above C source is as follows.

```

_enter_watch:
    LINK    #0              ; Secure a frame
    MOV     l:_lpmcr, #16; Clear TMD bit of LPMCR
    NOP                                ; 1st NOP
    NOP                                ; 2nd NOP
    UNLINK                                ; Release Frame
    RET                                 ; Finish enter_watch

*Since there is an UNLINK instruction, if other interrupt is occurred between NOP and RET
instruction, it will be failed. If there is a possibility that interrupts occur, LINK/UNLINK generation
need to be deterred via compiler configuration.

```

(2) Standby mode transition instruction can be written by `__asm` statements. Two NOP and JMP instructions are added right after Standby mode transition instruction.

```
__asm("      MOV I:_lpmcr,#H'58"); /* at SLP bit of LPMCR is set to "1" */
__asm("      NOP");                /* 1st NOP */
__asm("      NOP");                /* 2nd NOP */
__asm("      JMP  exit_sleep1");    /* JMP to exit_sleep1 label below*/
__asm("exit_sleep1      ");        /* Define exit_sleep1 label, destination of JMP */
```

The resulting assembler code of above C source is as follows

```
      MOV I:_lpmcr, # H'58      ;Unfold __asm sentence, set SLP bit of LPMCR
      NOP                      ; 1st NOP
      NOP                      ; 2nd NOP
      JMP  exit_sleep1        ; JMP to exit_sleep1
exit_sleep1                    ; Label of exit_sleep1
```

(3) Standby mode transition instruction can be written between `#pragma asm` and `#pragma endasm`. Two NOP and JMP instructions are added right after Standby mode transition instruction.

```
#pragma asm          /* Start with #pragma asm */
      MOV I:_lpmcr, #H'58 /* Set SLP bit of LPMCR */
      NOP              /* 1st NOP */
      NOP              /* 2nd NOP */
      JMP  exit_sleep  /* JMP to next exit_sleep */
exit_sleep          /* Destination of JMP */
#pragma endasm      /* Close with #pragma endasm */
```

The resulting assembler code of above C source is as follows

```
      MOV  I:_lpmcr, #H'58      ; #Unfold pragma asm, set SLP bit of LPMCR
      NOP                      ; 1st NOP
      NOP                      ; 2nd NOP
      JMP  exit_sleep          ; JMP to exist_sleep
exit_sleep                    ; Label of exit_sleep
```

6.5 Examples of programs not affected by Standby-Cancel problem

(1) There is an instruction that this problem is avoided as in Table 2.

MOV	I:LPMCR, H'10	; LPMCR: Set "0" to TMD
CLRB	PDR0:0	; Instruction which does not update the Queue counter (please see Table 2.)
MOV	A, 0300H	; Application

(2) There are two or more NOP instructions and a JMP*9, JMPP*9, JCTX instruction right after the Standby mode transition instruction.

MOV	I:LPMCR, H'10	; LPMCR: Set "0" to TMD
NOP		
NOP		
JMP	LABEL	;JMPP*9, JCTX are also same as JMP

*9: JMP, JMPP instructions, except JMP @(@PC+disp16),JMPP @(@PC+disp16)

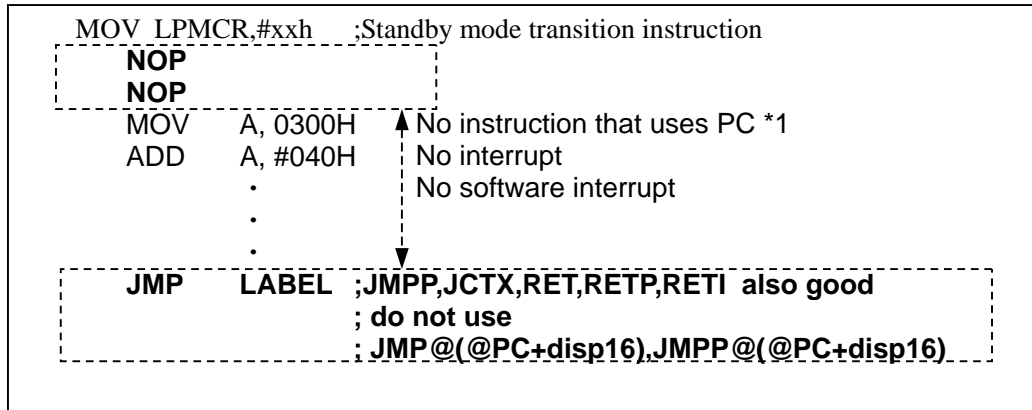
(3) There is a subroutine of Standby mode instructions and immediately following there are two or more NOP instructions, followed by a RET or RETP instruction.

MOV	I:LPMCR, H'98	; LPMCR: Set "1" to STP
NOP		
NOP		
RET		;RETP also as RET

(4) There is a subroutine of Standby mode instructions and immediately following there are two or more NOP instructions, followed by a RETI instruction.

MOV	I:LPMCR, H'58	; LPMCR: Set "1" to SLP
NOP		
NOP		
RETI		

(5) Next instruction is executed and there are two or more NOP instructions right after Standby mode transition instruction and there is JMP*11, JMPP*11, JCTX, RET, RETP, RETI instruction before any instruction*10 that it does use the program counter. There is no interrupt before execution of JMP*11, JMPP*11, JCTX, RET, RETP, RETI instruction.

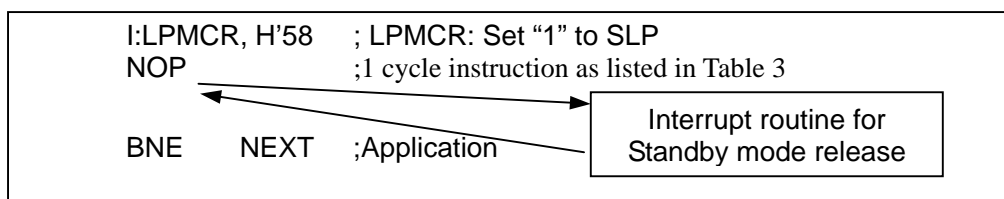


- *10: Instructions that use the PC
- Relative branch instructions

BZ/BEQ	BNT	CWBNE
BNZ/BNE	BLT	CBNE
BC/BLO	BGE	DBNZ
BNC/BHS	BLE	DWBNZ
BN	BGT	BBC
BP	BLS	BBS
BV	BHI	SBBS
BNV	BRA	
BT	CBNE	
 - CALL, CALLP, CALLV instructions , INT,INTP,INT9 instructions
 - Instructions that use PC indirect addressing as operand (@PC+disp16)
Example: MOV A, @PC+disp16

*11: JMP, JMPP instructions,
Except JMP @(@PC+disp16), JMPP @(@PC+disp16)

(6) An interrupt routine is executed by releasing Standby mode and there is an instruction that this problem is avoided as Table 3.



6.6 Examples of programs affected by Standby-Cancel problem

(1) There is one or no NOP right after the Standby mode transition instruction and no interrupt routine is executed by the Standby mode release interrupt.

Results: JMP instruction will not be recognized correctly.

MOV	I:LPMCR, H'10	; LPMCR: Set "0" to TMD
NOP		
JMP	LABEL	; MCU does not run correctly except NOP

(2) There is no instruction as listed in Table 3 right after Standby mode transition instruction even when an interrupt routine is executed by Standby mode release interrupt.

Results: RET instruction is not executed correctly.

MOV	I:LPMCR, H'10	; LPMCR: Set "0" to TMD
RET		; MCU does not run correctly except instruction as listed in Table 3

(3) There is an instruction*10 that uses PC before a JMP*11, JMPP*11, JCTX, RET, RETP or RETI instruction and an interrupt routine is not executed by the Standby mode release interrupt. Even when there are two NOP instructions right after Standby mode transition instruction the failure will show up.

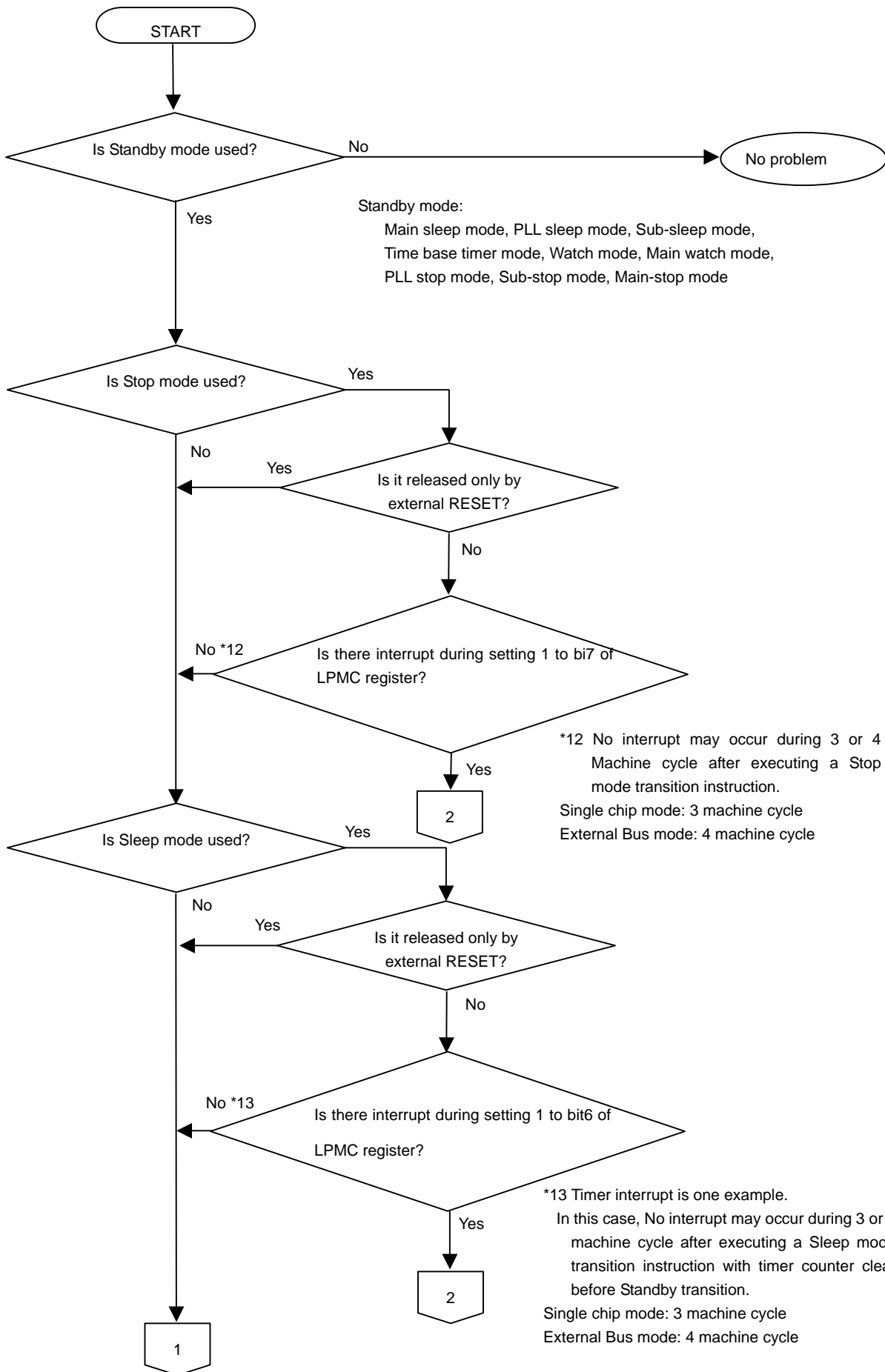
Results: Branch address becomes wrong for BBC instruction.

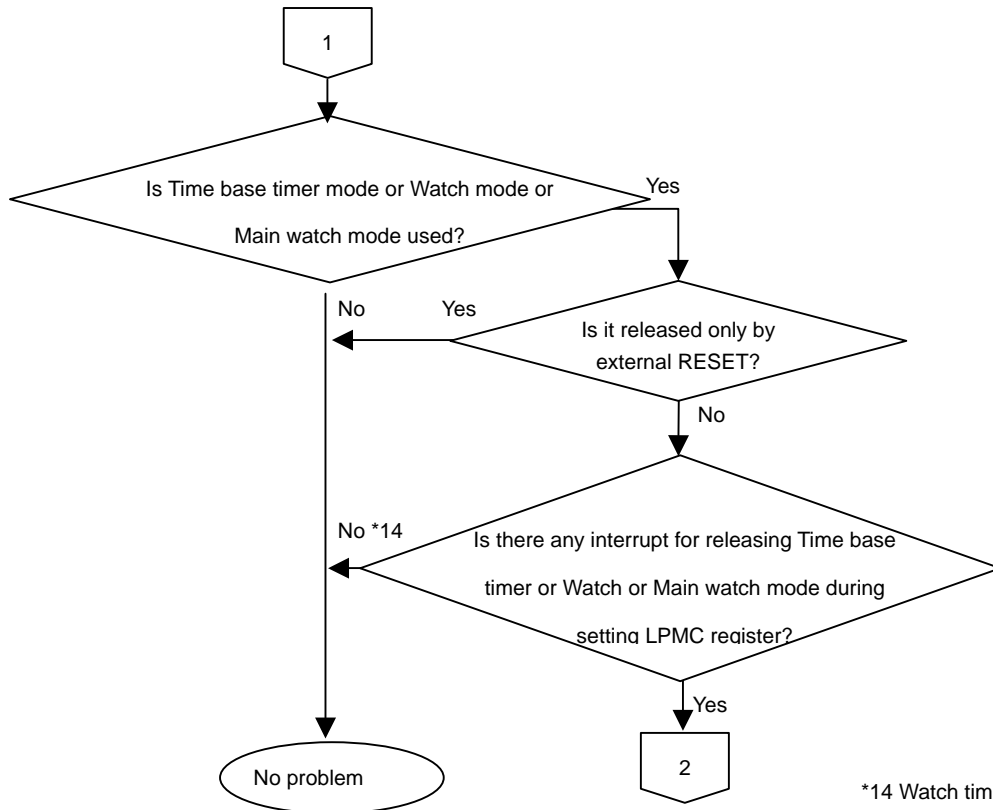
MOV	I:LPMCR, H'98	; LPMCR: Set "1" to STP
NOP		
NOP		
BBC	CKSCR:SCM, LABEL	; MCU will do a wrong execution with an instruction*10 that uses PC
RET		

Results: Return address becomes wrong for CALL instruction.

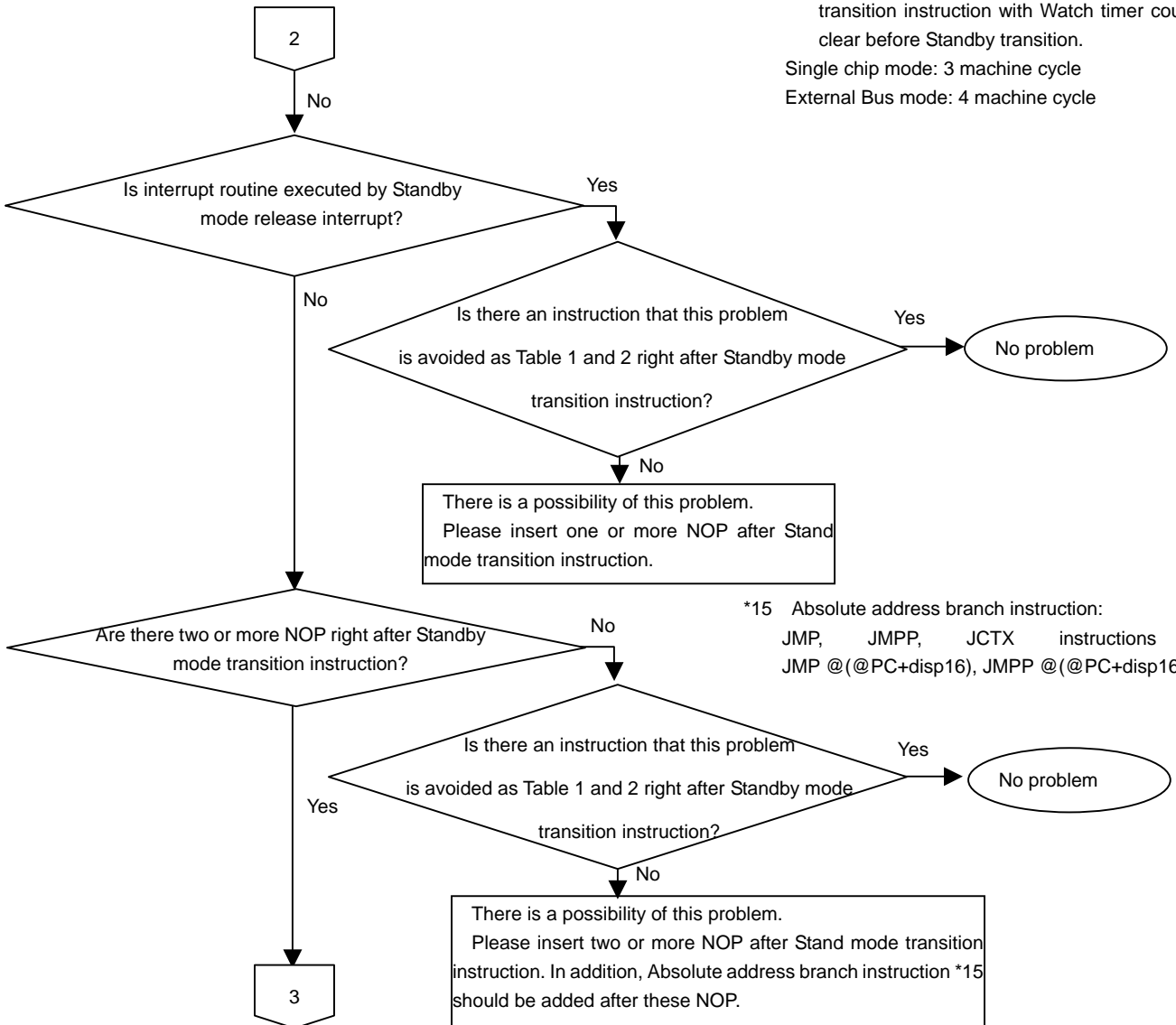
MOV	I:LPMCR, H'98	
NOP		
NOP		
MOV	A, @R0	
CALL	SUBROUTINE	; MCU will do a wrong execution with an instruction*10 that uses PC
RET		

6.7 Flowchart to decide whether Standby-Cancel problem affects existing software





*14 Watch timer interrupt is one example.
 In this case, No interrupt may occur during 3 or 4 machine cycle after executing a Sleep mode transition instruction with Watch timer counter clear before Standby transition.
 Single chip mode: 3 machine cycle
 External Bus mode: 4 machine cycle



*15 Absolute address branch instruction:
 JMP, JMPP, JCTX instructions except
 JMP @(@PC+disp16), JMPP @(@PC+disp16)

