

UTMC Errata Sheet

UT80CXX196KD MicroController ALU Anomaly

1.0 Overview:

The “A” and “B” revisions of the UT80CXX196KD (identified by either JD02A or JD02B on the lid marking) exhibits an ALU anomaly when some program status flags are not updated according to the instruction definition. Table 1, describes the anomalous instructions in conjunction with their at-risk program status flags (Note: The program status flags are located in the upper byte of the Program Status Word (PSW)):

Table 1: Anomalous Instructions

Instruction	Affected Program Status Flags	Comment
ADDC	V (PSW bit 5)	The overflow (V) flag is not updated following an ADDC instruction.
AND	V (PSW bit 5)	The overflow (V) flag will not be cleared as required by the AND instruction.
OR	V (PSW bit 5)	The overflow (V) flag will not be cleared as required by the OR instruction.
XOR	V (PSW bit 5)	The overflow (V) flag will not be cleared as required by the XOR instruction.
CLR	Z (PSW bit 7) N (PSW bit 6) V (PSW bit 5) C (PSW bit 3)	The zero (Z), negative (N), overflow (V), and carry (C) flags are not updated. Following the execution of a CLR instruction, update the flags to the following values: Z = 1, N = 0, V = 0, and C = 0.
CLRB	Z (PSW bit 7) N (PSW bit 6) V (PSW bit 5) C (PSW bit 3)	The zero (Z), negative (N), overflow (V), and carry (C) flags are not updated. Following the execution of a CLRB instruction, update the flags to the following values: Z = 1, N = 0, V = 0, and C = 0.

Incorrect flag updating following the execution of an AND, OR, XOR, CLR, or CLRB instruction does not affect a user’s application code because a conditional jump, based on the erroneous flags, does not occur following the execution of an at-risk instruction. When using an AND, OR, XOR, CLR, or CLRB instruction, the status of the erroneous flags is expected to be a known value: therefore, no condition would need to be evaluated in order to execute a jump instruction. However, the ADDC instruction may impact a user’s application code if a JV or JNV instruction follows the ADDC instruction without the proper update of the overflow (V) flag.

UTMC has identified several software solutions to the ADDC anomaly which are discussed in section 2.0 below. Revision C of the JD02 die is underway. Prototypes will be available in November 1999 with production readiness in December 1999. Until JD02 Revision C die are ready for customer use, UTMC recommends the user’s application code include the software solutions to the ADDC anomaly listed in this document.

2.0 ADDC Anomaly Solutions:

Because jump instructions using the program status flags can occur following the execution of an ADDC instruction, the user should consider one of the following work-arounds to temporarily solve the overflow (V) flag update anomaly.

2.0.1 ADDC Solution #1:

The first solution to the ADDC anomaly is to ensure that a JV or JNV instruction is not used to evaluate the overflow flag of the ADDC instruction. If the software is written in C/C++ and compiled into assembly language, this solution may be difficult to control.

UTMC Errata Sheet

2.0.2 ADDC Solution #2:

Do not use signed arithmetic. If the application code only uses unsigned arithmetic no JV or JNV instructions will be required to evaluate a jump based on an overflow condition. Instead, only JC and JNC instructions should be used for unsigned arithmetic.

2.0.3 ADDC Solution #3:

The last software solution is to avoid using the ADDC instruction. Again, this is difficult to control if the applications code is written in C/C++ and compiled into assembly. Furthermore, the ADDC instruction should be replaced with a subroutine that produces the same result as the ADDC instruction. However, this solution requires some instruction overhead to emulate an ADDC instruction. The following example provides a software subroutine that can be used to perform an ADDC instruction:

```
ADDC_emulation:      ;Begin ADDC emulation subroutine

    JNC no_carry     ;Check if the carry flag is set, jump if not

    ADD R0, #1       ;If the carry flag was set add one immediate to the first reg

    JC  carry_out    ;Check if adding one caused a carry condition

no_carry:            ;Enter the no_carry subroutine

    ADD R0, R2       ;Add the original registers together
    SJMP done        ;The ADDC emulation is complete

carry_out:           ;Enter the carry_out subroutine

    LD  R0, R2       ;If adding one causes a carry condition, then R0 would...
                    ;...end up with the contents of R2
                    ;The LD instruction does not affect the carry (C) or overflow (V) flags...
                    ;...and the corresponding flag should still be set.

done:                ;Finish the subroutine

    RET              ;Return to the calling location.
```

Lastly, in place of all ADDC instructions, the user would replace the 3, 4, or 5 byte ADDC instruction with a 2 or 3 byte SCALL or LCALL respectively. The SCALL or LCALL instruction would point to the beginning of the ADDC_emulation subroutine.