

CS+CC で確認のためのプロジェクトを作成し、TESSERA 社の RL78/G14 Stick で確認しました。

確認のために、INTP0～INTP4 の優先順位を以下のように設定しておきます。

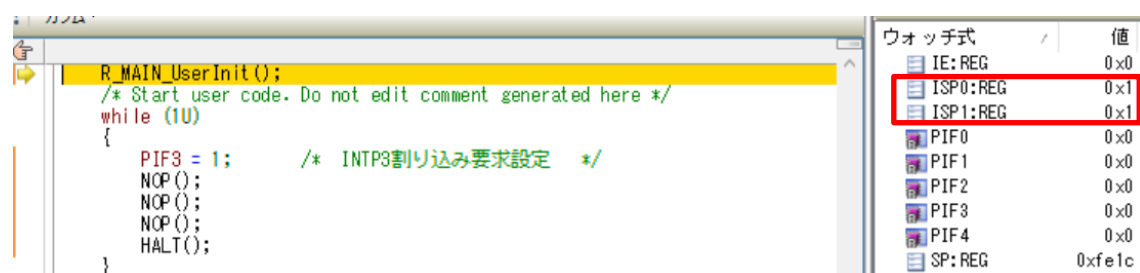
外部割り込み		キー入力割り込み	
- INTP0 設定			
<input checked="" type="checkbox"/> INTP0	有効エッジ	立下りエッジ	優先順位 高
- INTP1 設定			
<input checked="" type="checkbox"/> INTP1	有効エッジ	立下りエッジ	優先順位 レベル1
- INTP2 設定			
<input checked="" type="checkbox"/> INTP2	有効エッジ	立下りエッジ	優先順位 レベル2
- INTP3 設定			
<input checked="" type="checkbox"/> INTP3	有効エッジ	立下りエッジ	優先順位 低
- INTP4 設定			
<input checked="" type="checkbox"/> INTP4	有効エッジ	立下りエッジ	優先順位 高

その他の設定はプロジェクトを参照してください。

R_MAIN_UserInit 関数では、INTP0～INTP3 の割り込みマスクを解除しておきます。

```
void R_MAIN_UserInit(void)
{
    /* Start user code. Do not edit comment generated here */
    PMK0 = 0; /* 割り込みマスク解除 */
    PMK1 = 0;
    PMK2 = 0;
    PMK3 = 0;
    EI(); /* ベクタ割り込み許可 */
    /* End user code. Do not edit comment generated here */
}
```

その上で、main 関数では INTP3 の割り込み要求をプログラムで設定します。初期状態では、以下のように ISP1,ISP0 は 11 になっています。



各割り込み処理(r_cg_intc_user.c)の各割り込みにはブレークポイントを設定しておき、プログラムをブレークポイント付きで実行します。すると、当然ながら、INTP3 割り込みの処理部分でブレークします。INTP3 は優先順位が低(レベル 3)なので、ISP1,ISP0 は 10 になります。ここで、多重割り込みのために EI()を実行し、INTP3 と INTP2 の割り込み要求をプログラムで設定します。(INTP3 の方を先に設定して、NOP()で割り込みを受け付ける時間を確保しておきます。)

```

static void __near r_intc3_interrupt(void)
{
    /* Start user code. Do not edit comment generated here */
    EI();
    PIF3 = 1;
    NOP();
    NOP();
    NOP();
    PIF2 = 1; /* INTP2割り込み要求設定 */
    NOP();
}

```

ウォッチ式	値
IE: REG	0x1
ISP0: REG	0x0
ISP1: REG	0x1
PIF0	0x0
PIF1	0x0
PIF2	0x0
PIF3	0x1
PIF4	0x0
SP: REG	0xfe16

実行を再開すると、INTIP2 割り込み処理に分岐し、ISP1,ISP0 は 01 に変化しています。さらに、PIF3 をセットしていますが、INTP3 処理には分岐していません。

```

static void __near r_intc2_interrupt(void)
{
    /* Start user code. Do not edit comment generated here */
    EI();
    PIF2 = 1;
    NOP();
    NOP();
    NOP();
    PIF1 = 1; /* INTP1割り込み要求設定 */
    NOP();
}

```

ウォッチ式	値
IE: REG	0x1
ISP0: REG	0x1
ISP1: REG	0x0
PIF0	0x0
PIF1	0x0
PIF2	0x1
PIF3	0x1
PIF4	0x0
SP: REG	0xfe10

さらに、実行を再開します。今度は INTIP1 割り込み処理に分岐し、ISP1,ISP0 は 00 に変化しています

```

static void __near r_intc1_interrupt(void)
{
    /* Start user code. Do not edit comment generated here */
    EI();
    PIF1 = 1;
    NOP();
    NOP();
    NOP();
    PIF0 = 1; /* INTP0割り込み要求設定 */
    NOP();
}

```

ウォッチ式	値
IE: REG	0x1
ISP0: REG	0x0
ISP1: REG	0x0
PIF0	0x0
PIF1	0x1
PIF2	0x1
PIF3	0x1
PIF4	0x0
SP: REG	0xfe0a

さらに、実行を再開します。今度は INTIP0 割り込み処理に分岐します。ISP1,ISP0 は 00 のままです。

```

static void __near r_intc0_interrupt(void)
{
    /* Start user code. Do not edit comment generated here */
    EI();
    NOP();
    PMK4 = 0;
    PIF4 = 1;
    NOP();
    NOP();
    NOP();
    HALT();
}

```

ウォッチ式	値
IE: REG	0x1
ISP0: REG	0x0
ISP1: REG	0x0
PIF0	0x0
PIF1	0x1
PIF2	0x1
PIF3	0x1
PIF4	0x0
SP: REG	0xfe06

さらに、INTP4 割り込み(優先順位最高)をセットします。これで、実行を再開します。

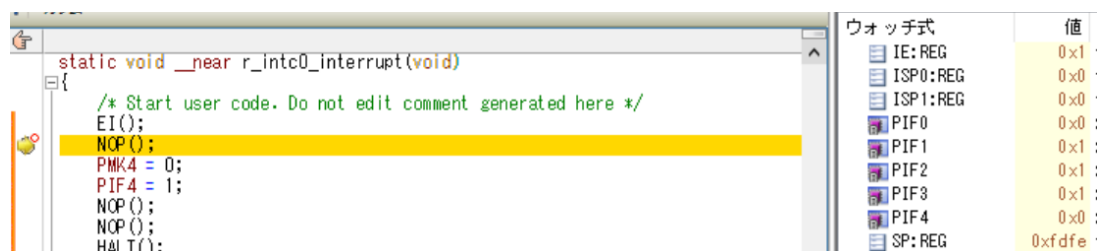
```

static void __near r_intc4_interrupt(void)
{
    /* Start user code. Do not edit comment generated here */
    EI();
    PIF0 = 1;
    NOP();
    NOP();
    NOP();
    NOP();
    /* End user code. Do not edit comment generated here */
}

```

すると INTP4 の割り込み処理が起動していることが分かります。

ここでは, INTP0 割り込みをセットしているので, 実行を再開すると, INTP0 割り込み処理に移るはずですが。実行結果を下に示します。このように, 優先順位最高の割り込みは EI()を実行するとどんどん割り込みを処理するのが分かります。



The screenshot shows an IDE with two panels. The left panel displays assembly code for a function named `__near r_intc0_interrupt(void)`. The code includes a comment `/* Start user code. Do not edit comment generated here */` and several instructions: `EI();`, `NOP();`, `PMK4 = 0;`, `PIF4 = 1;`, `NOP();`, `NOP();`, and `HALT();`. The `NOP();` instruction following `EI();` is highlighted in yellow. The right panel is a 'ウォッチ式' (Watch) window showing a list of registers and their values:

ウォッチ式	値
IE: REG	0x1
ISP0: REG	0x0
ISP1: REG	0x0
PIF0	0x0
PIF1	0x1
PIF2	0x1
PIF3	0x1
PIF4	0x0
SP: REG	0xfdfc

ここまでで処理は, すべて多重割り込みで処理しているので, SP の値がどんどん小さくなっているのも確認できます。