


|   |  |
|---|--|
|  | <p>Министерство науки и высшего образования Российской Федерации<br/>         Мытищинский филиал<br/>         Федерального государственного автономного образовательного учреждения<br/>         высшего образования<br/>         «Московский государственный технический университет<br/>         имени Н.Э. Баумана<br/>         (национальный исследовательский университет)»<br/>         (МФ МГТУ им. Н.Э. Баумана)</p> |
|---|--|

ФАКУЛЬТЕТ КОСМИЧЕСКИЙ

КАФЕДРА К-3

## отчет

### *К ЛАБОРАТОРНОЙ РАБОТЕ*

### по ДИСЦИПЛИНЕ

### «ОРГАНИЗАЦИЯ ЭВМ И СИСТЕМ»

Студент К3-76Б  
(Группа)

Студент К3-76Б  
(Группа)

Преподаватель

Чернов В.Д.  
(И.О.Фамилия)

Сериков Д.Е.  
(И.О.Фамилия)

Н.В.Ефремов  
(И.О.Фамилия)

2025 г.

## **Цель работы:**

Изучить возможные способы ввода/вывода информации в процессорной системе.

## **Исходные файлы лабораторной работы**

В приложении приводится исходный код процедуры `hex_display` с подробными комментариями. Процедура отображает содержимое регистра `r4` в шестнадцатеричном формате на 7-сегментных индикаторах.

Кроме того там находится исходный файл тестовой программы *`interrupt_example.s`*, демонстрирующей прерывания процессорной системы. В ней используются прерывания от таймера и кнопок.

Обработчик исключений, задачей которого является определение причины прерывания и вызов соответствующей процедуры, содержится в файле *`exception_handler.s`* (Листинг 10).

Обработчики прерываний от таймера и кнопок содержатся в файлах *`interval_timer.s`* и *`pushbutton_ISR.s`* соответственно.

# Выполнение заданий лабораторной работы

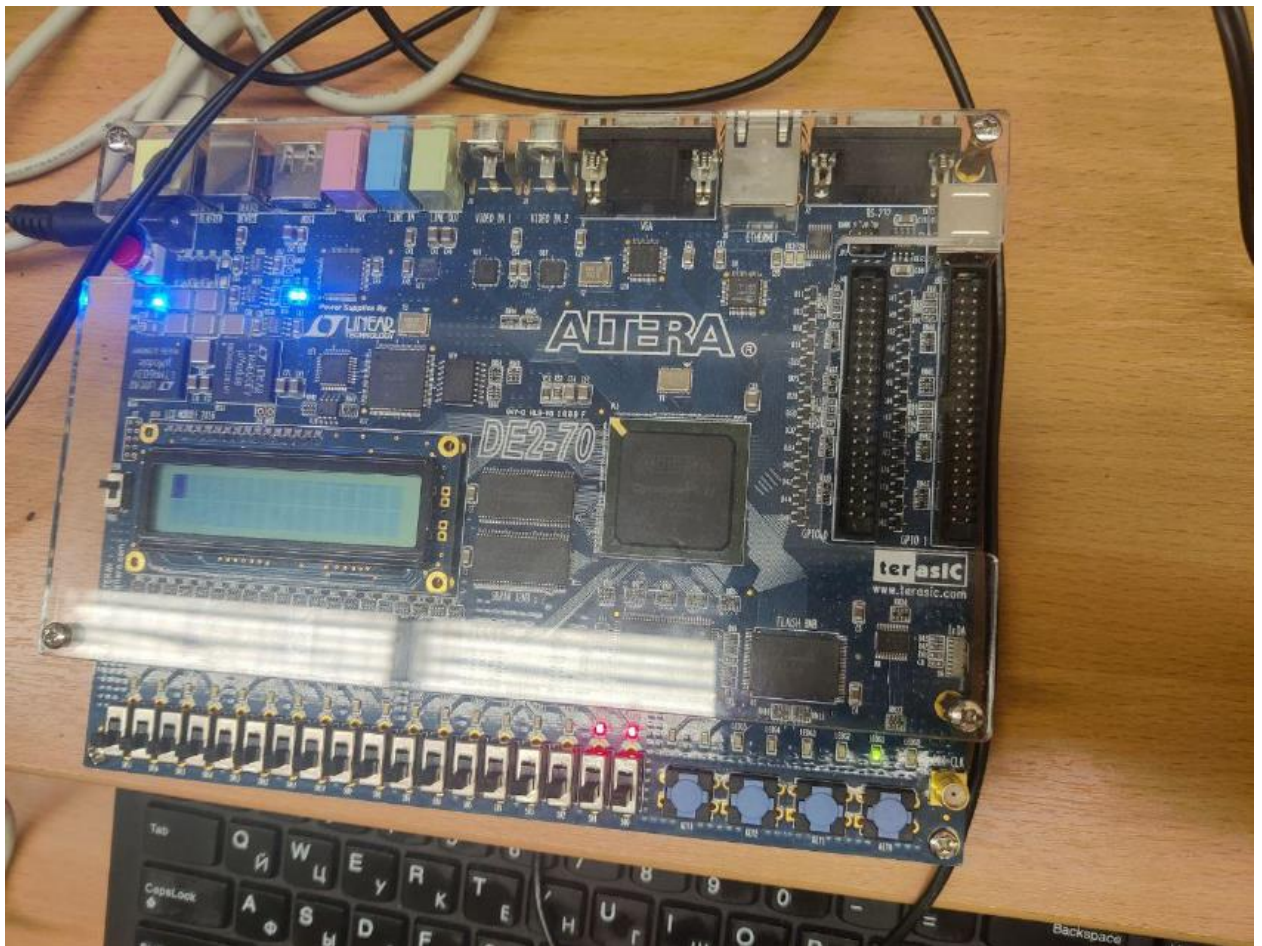
## Часть 1. Программно управляемый ввод с переключателей и вывод на светодиоды

Использовали 8 правых переключателей SW 7.0 для задания вводимого числа и 8 зеленых светодиодов LEDG 7.0 для его отображения. Использовали 18 красных светодиодов LEDR 17.0 для отображения младших разрядов вычисленной суммы. Все эти устройства подключаются через параллельные порты к мультимедийной процессорной системе.

- Написали программу на языке ассемблер, которая вначале обнуляет сумму, затем читает 8-разрядное значение (число со знаком в дополнительном коде), задаваемое переключателями SW 7.0, отображает прочитанное значение на зеленых светодиодах, выполняет сложение введенного числа с накопленной суммой, выводит сумму на красные светодиоды. Осуществляет переход на ввод очередного числа, выполняя бесконечный цикл.
- Создали новую папку *lab7\_part* и поместили в неё подготовленную программу.
- Разместили сегменты кода и данных со смещением адреса на 0x1000. Область в начале оперативной памяти понадобится в последующем для размещения обработчиков сброса и прерываний процессорной системы. Скомпилировали и загрузили нашу программу.
- Выполнили программу по шагам, задавая различные значения вводимых чисел.
- Поставили контрольную точку по адресу безусловного перехода на ввод очередного числа (br met). Выполнили рестарт программы. Чтобы убедиться в работоспособности, мы каждый раз задавали новое число и нажимали кнопку продолжить.:

### Листинг программы:

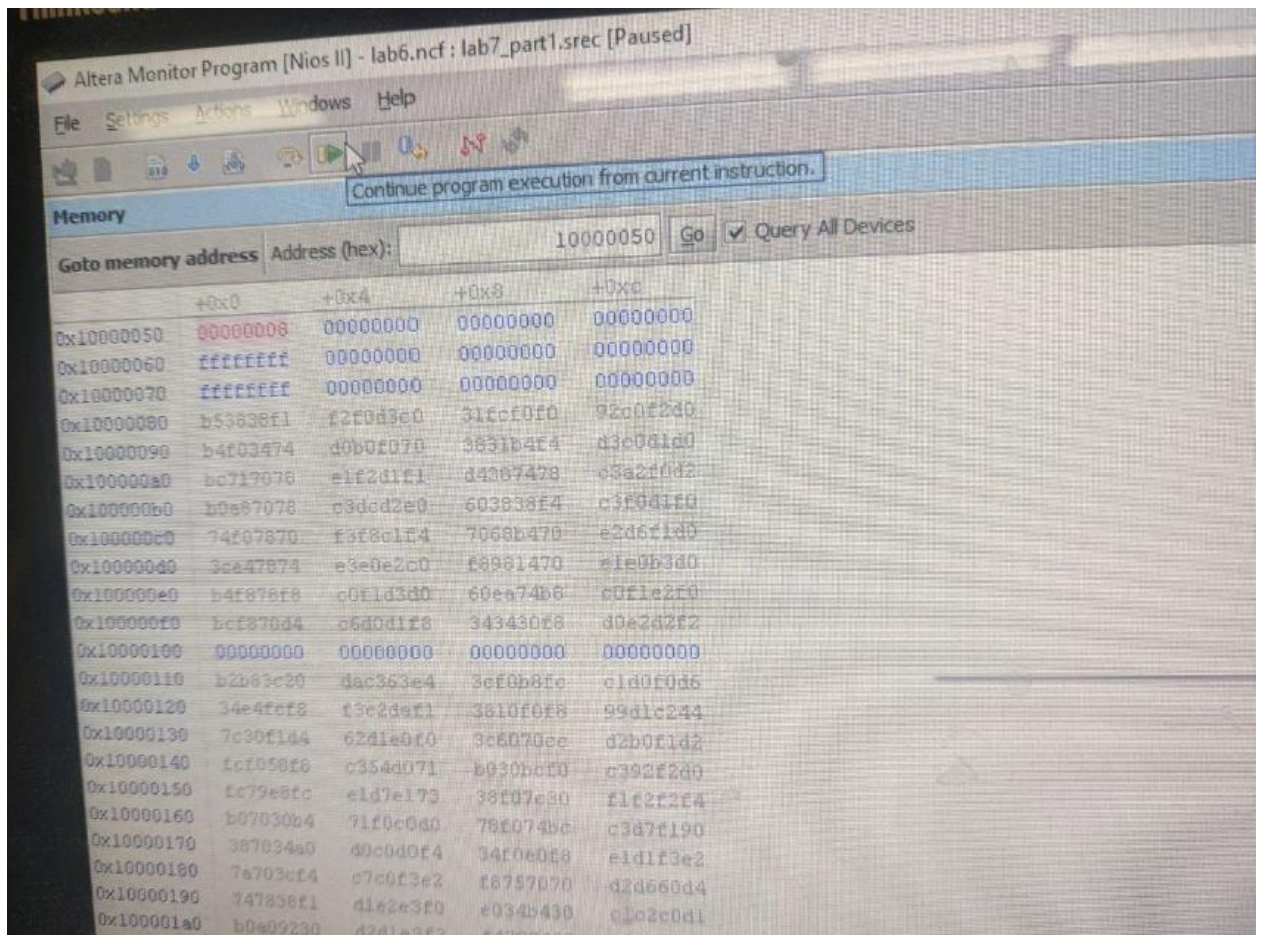
```
.equ TIMER_BASE_ADDR, 0x10002000
.text
.global _start
_start:
movia r10, TIMER_BASE_ADDR
/* записываем максимальное значение в counter start */
movia r1, 0xffff
movia r2, 0x7fff
sthio r1, 0x8(r10)
sthio r2, 0xc(r10)
/* записываем в бит start единицу */
movia r4, 0b00000100
stbio r4, 4(r10)
/* делаем запись любого числа в младшую часть регистра snapshot для отображения текущего значения таймера*/
movia r4, 1
sthio r4, 0x10(r10)
/* получаем значения snapshot */
ldhuio r3, 0x10(r10)
ldhuio r4, 0x14(r10)
/* объединяем младшие и старшие части */
slli r5, r2, 16
or r5, r5, r1
slli r6, r4, 16
or r6, r6, r3
/* вычисляем пройденное время в миллисекундах */
sub r7, r5, r6
movia r15, 50000
div r8, r7, r15
met:
br met
.end
```



— Удаляем контрольную точку и вновь запускаем программу. Наблюдаем загорание всех используемых в программе красных светодиодов (ledr0 – ledr7). Это происходит т.к. мы используем бесконечный цикл и в автоматическом режиме, из-за скорости выполнения программы мы не можем корректно отследить изменение красных светодиодов.

## Часть 2. Ввод информации с переключателей с опросом готовности

Остановили выполнение программы. Наблюдали состояние регистра данных и регистра захвата фронтов (edge-capture) PIO, соединенного с кнопками на стенде. Для этого использовали вкладку *Memory* приложения IMP. Включили опцию *Query all devices*, используя соответствующее окно. Сначала нажали и удерживали кнопки KEY3 и KEY1, затем наблюдали значение регистра 0x10000050 равное 0x0 в окне *Memory*, затем нажали кнопку *Refresh* в окне AMP и состояние регистра изменилось на 0xA (1010 в бинарном). Отпустили кнопки, изменения регистров не наблюдали, после нажатия *Refresh* и наблюдаем обнуление значения регистра 0x10000050 и взводится соответствующий бит в регистре 0x1000005C, который был регистре 0x10000050 на момент зажатия.



Модифицировали программу из предыдущей части таким образом, чтобы ввод очередного числа осуществлялся только после отпускания кнопки KEY3 на стенде. Используем соответствующий разряд регистра *edge-capture* соответствующего PIO как сигнал готовности. Программа опрашивает его непрерывно до тех пор, пока он не установится. После ввода числа не забыли снять сигнал готовности, записав нулевое значение в регистр *edge-capture*.

Листинг программы:

```
.equ TIMER_BASE_ADDR, 0x10002000
.text
.global _start
_start:
movia r10, TIMER_BASE_ADDR
/* очистка регистра состояния */
sthio r0, 0(r10)
/* записываем максимальное значение в counter start */
movia r1, 0xffff
movia r2, 0x7fff
sthio r1, 0x8(r10)
sthio r2, 0xc(r10)
/* записываем в бит start единицу */
movia r4, 0b00000100
stbio r4, 4(r10)
subi r12, r12, 1
/* записываем в бит stop единицу */
movia r4, 0b00001000
stbio r4, 4(r10)
/* делаем запись любого числа в младшую часть регистра snapshot для отображения текущего значения таймера */
movia r4, 1
sthio r4, 0x10(r10)
/* получаем значения snapshot */
ldhuio r3, 0x10(r10)
ldhuio r4, 0x14(r10)
/* объединяем младшие и старшие части */
slli r5, r2, 16
or r5, r5, r1
```

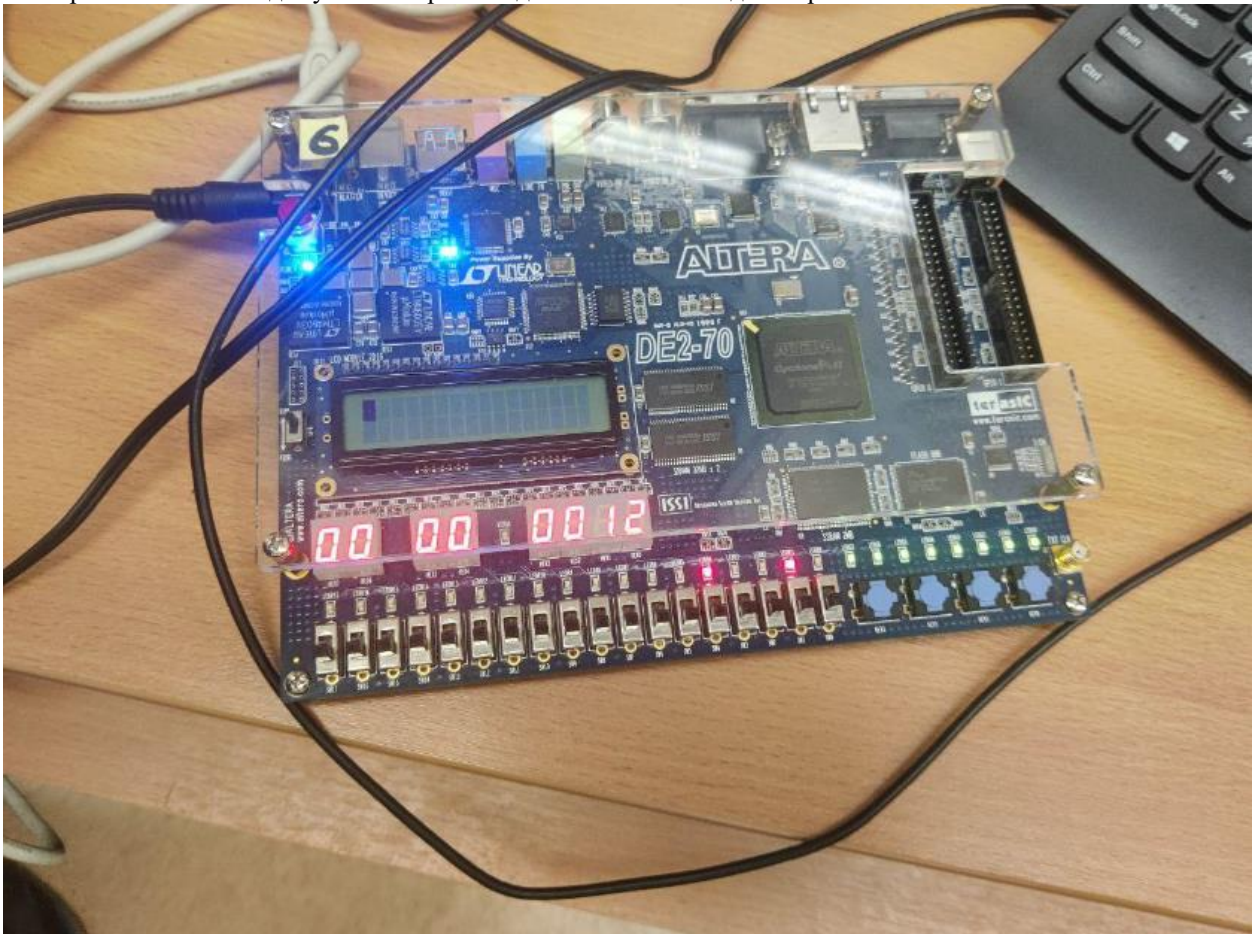
```

slli r6, r4, 16
or r6, r6, r3
/* вычисляем пройденное время в миллисекундах */
sub r7, r5, r6
movia r15, 50000
div r8, r7, r15
met:
br met
.end

```

- Добавили в проект файл с процедурой `hex_display` из приложения, используя команду Program Setting из меню Settings программы IMP. Процедура отображает содержимое 32-разрядного регистра процессора на 7-сегментных индикаторах `HEX7-HEX0` в виде шестнадцатеричного числа.
- Модифицировали программу из предыдущей части таким образом, чтобы накопленная сумма дополнительно отображалась на индикаторах `HEX7-HEX0`. Для этого мы добавили команду `call hex_display` в подпрограмму "s".

Фото работы нового кода сумма теперь выводится и на HEX индикаторах:



### Часть 3. Ввод информации с переключателей в режиме прерывания

Выполнили задачу из предыдущей части. Использовали вместо опроса готовности устройства ввода формирование прерывания, которое инициируется нажатием кнопки `KEY3` на стенде. Основная программа устанавливает все регистры, необходимые для формирования прерывания и выполняет бесконечный цикл. Программа обслуживания прерывания вводит число, вычисляет сумму и отображает ее на светодиодах и индикаторах.

#### Листинг программы:

```

//interval_timer
.equ LEDS_BASE, 0x10000000
.include "key_codes.s" /* includes EQU for KEY1, KEY2 */

```

```

.global INTERVAL_TIMER_ISR
INTERVAL_TIMER_ISR:
subi sp, sp, 12 /* reserve space on the stack */
stw ra, 0(sp)
stw r5, 4(sp)
stw r16, 8(sp) movia r10, 0x10002000
sthio r0, 0(r10) /* clear the interrupt */
movia r16, LEDS_BASE
movia r5, 0x3ffff
stwio r5, 0(r16) /* Выводим результат на светодиоды */
br END_INTERVAL_TIMER_ISR
END_INTERVAL_TIMER_ISR:
ldw ra, 0(sp) /* Restore all used register to previous */
ldw r5, 4(sp)
ldw r16, 8(sp)
addi sp, sp, 12 /* release the reserved space on the stack */
ret
.end
//exception_handler
.section .reset, "ax"
movia r2, _start
jmp r2 /* branch to main program */
.section .exceptions, "ax"
.global EXCEPTION_HANDLER
EXCEPTION_HANDLER:
subi sp, sp, 16 /* make room on the stack */
stw et, 0(sp)
rdctl et, ctl4
beq et, r0, SKIP_EA_DEC /* interrupt is not external */
subi ea, ea, 4 /* must decrement ea by one instruction */
/* for external interrupts, so that the */
/* interrupted instruction will be run after eret */
SKIP_EA_DEC:
stw ea, 4(sp) /* save all used registers on the Stack */
stw ra, 8(sp) /* needed if call inst is used */
stw r22, 12(sp)
rdctl et, ctl4
bne et, r0, CHECK_LEVEL_0 /* exception is an external interrupt */
NOT_EI: /* exception must be unimplemented instruction or TRAP */
br END_ISR /* instruction. This code does not handle those cases */
CHECK_LEVEL_0: /* interval timer is interrupt level 0 */
stw r5, 4(sp)

```

Проверили экспериментально действия программы, выполняемые после формирования прерывания. Для этого поставили контрольную точку по адресу обработчика прерываний (0x20). После достижения контрольной точки выполнили программу по шагам.

| Goto instruction |          |        | Address (hex) or symbol name: 0 |  | Go |  | Hide |  | REGISTERS |            |
|------------------|----------|--------|---------------------------------|--|----|--|------|--|-----------|------------|
|                  |          |        |                                 |  |    |  |      |  | Reg       | Value      |
| 0x00000000       | 00800034 | orhi   | r2, zero, 0x0                   |  |    |  |      |  | pc        | 0x00000020 |
| 0x00000004       | 10840804 | addi   | r2, r2, 0x1020                  |  |    |  |      |  | zero      | 0x00000000 |
| 0x00000008       | 1000683a | jmp    | r2                              |  |    |  |      |  | r1        | 0x00000000 |
| 0x0000000c       | 2155eb4d | sth    | r5, 22445(r4)                   |  |    |  |      |  | r2        | 0x00001020 |
| 0x00000010       | 51cef814 | ori    | r7, r10, 0x3be0                 |  |    |  |      |  | r3        | 0x00000001 |
| 0x00000014       | d121742b | ldhuio | r4, -31280(cp)                  |  |    |  |      |  | r4        | 0x00000000 |
| 0x00000018       | 6cb0d90c | andi   | r10, r13, 0xc364                |  |    |  |      |  | r5        | 0x00000000 |
| 0x0000001c       | 5e05600b | ldhu   | et, 5504(r11)                   |  |    |  |      |  | r6        | 0x00000000 |
|                  |          |        |                                 |  |    |  |      |  | r7        | 0x00000000 |
|                  |          |        |                                 |  |    |  |      |  | r8        | 0x00000000 |
|                  |          |        |                                 |  |    |  |      |  | r9        | 0x00000000 |
|                  |          |        |                                 |  |    |  |      |  | r10       | 0x10000058 |
|                  |          |        |                                 |  |    |  |      |  | r11       | 0x00000000 |
|                  |          |        |                                 |  |    |  |      |  | r12       | 0x00000000 |
| 0x00000020       | d9800015 | stw    | r6, 0(sp)                       |  |    |  |      |  |           |            |

После нажатия кнопки Key3 программа перешла на контрольную точку, где начинается обработка прерывания, в результате чего вычисляется сумма.

Чтобы визуализировать работу основной программы, реализовали в ней вывод на LCD дисплей нашей фамилии, имени и отчества в режиме бегущей строки. Заготовку программы взяли из 3 лабораторной работы.

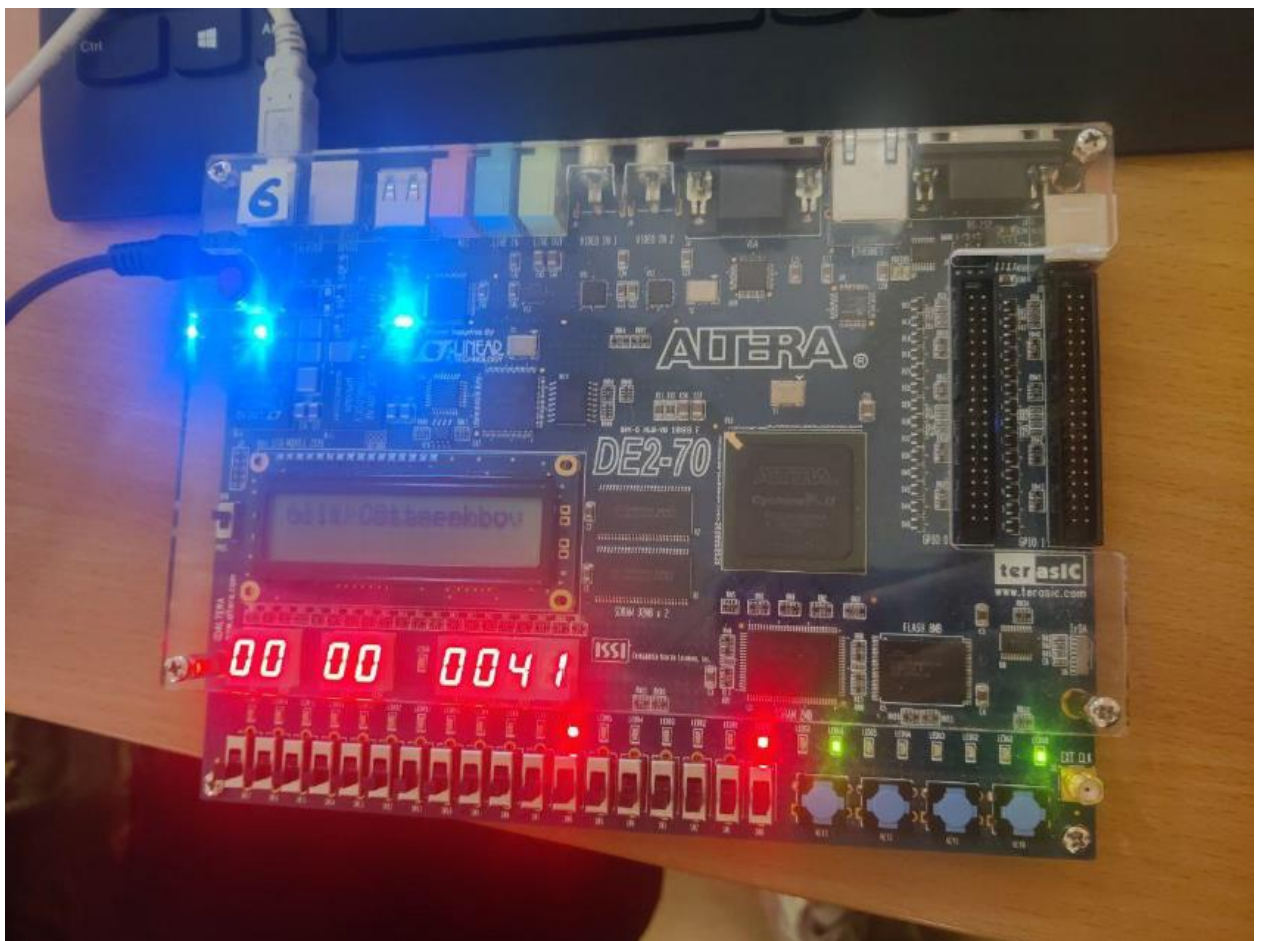
Листинг программы:

```
/exception_handler
.section .reset, "ax"
movia r2, _start
jmp r2 /* branch to main program */
.section .exceptions, "ax"
.global EXCEPTION_HANDLER
EXCEPTION_HANDLER:
subi sp, sp, 16 /* make room on the stack */
stw et, 0(sp)
rdctl et, ctl4
subi ea, ea, 4 /* must decrement ea by one instruction */
/* for external interrupts, so that the */
/* interrupted instruction will be run after eret */
SKIP_EA_DEC:
stw ea, 4(sp) /* save all used registers on the Stack */
stw ra, 8(sp) /* needed if call inst is used */
stw r22, 12(sp)
rdctl et, ctl4
bne et, r0, CHECK_LEVEL_0 /* exception is an external interrupt */
NOT_EI: /* exception must be unimplemented instruction or TRAP */
br END_ISR /* instruction. This code does not handle those cases */
CHECK_LEVEL_0: /* interval timer is interrupt level 0 */
andi r22, et, 0b1
beq r22, r0, END_ISR /* при добавлении нового прерывания поменять метку */
call INTERVAL_TIMER_ISR
br END_ISR
END_ISR:
ldw et, 0(sp) /* restore all used register to previous values */
ldw ea, 4(sp)
ldw ra, 8(sp) /* needed if call inst is used */
ldw r22, 12(sp)
addi sp, sp, 16
eret
.end
//interval_timer
.equ LEDS_BASE, 0x10000000
.include "key_codes.s" /* includes EQU for KEY1, KEY2 */
.global INTERVAL_TIMER_ISR
INTERVAL_TIMER_ISR:
subi sp, sp, 16 /* reserve space on the stack */
stw ra, 0(sp)
stw r4, 4(sp)
stw r16, 8(sp)
stw r10, 12(sp)
movia r10, 0x10002000 /* interval timer base address */
```

```

sthio r0, 0(r10) /* clear the interrupt */
movia r16, seconds
ldb r4, 0(r16)
addi r4, r4, 1
stb r4, 0(r16)
call hex_display
call led_display
br END_INTERVAL_TIMER_ISR
END_INTERVAL_TIMER_ISR:
ldw ra, 0(sp) /* Restore all used register to previous */
ldw r4, 4(sp)
ldw r16, 8(sp)
ldw r10, 12(sp)
addi sp, sp, 16 /* release the reserved space on the stack */
ret
.data
seconds:
.byte 0

```



## Часть 4. Реализация приоритетных прерываний

Модифицировали программу из предыдущей части таким образом, чтобы по кнопке *KEY3* осуществлялось сложение нового числа с суммой, по кнопке *KEY2* - *вычитание нового числа из суммы*, по кнопке *KEY1* - обнуление суммы. Причем реализовали приоритетность прерываний в соответствии с указанной последовательностью.

Листинг программы:

```
//pushbutton
#include "key_codes.s" /* defines values for KEY1, KEY2, KEY3 */
.extern minutes
.extern hours
.extern seconds
.global PUSHBUTTON_ISR
PUSHBUTTON_ISR:
subi sp, sp, 40 /* reserve space on the stack */
stw ra, 0(sp)
stw r11, 8(sp)
stw r12, 12(sp)
stw r13, 16(sp)
stw r16, 20(sp)
stw r17, 24(sp)
stw r4, 28(sp)
stw r5, 32(sp)
stw r6, 36(sp)
movia r10, pushbutton_addr /* base address of pushbutton KEY parallel port */
ldwio r11, 0xC(r10) /* read edge capture register */
stwio r0, 0xC(r10) /* clear the interrupt */
movia r16, timer_addr
ldw r17, 4(r16)
andi r17, r17, 0b100
CHECK_KEY1:
andi r13, r11, 0b0010 /* check KEY1 */
beq r13, zero, CHECK_KEY2
beq r17, r0, start_timer
movi r10, 0b1000
sthio r10, 4(r16)
br END_PUSHBUTTON_ISR
start_timer:
movi r10, 0b0111 /* START = 1, CONT = 1, ITO = 1 */
sthio r10, 4(r16)
br END_PUSHBUTTON_ISR
CHECK_KEY2:
andi r13, r11, 0b0100 /* check KEY2 */
beq r13, zero, DO_KEY3
bne r17, r0, END_PUSHBUTTON_ISR
movia r12, minutes
movia r13, switch_addr /* SW slider switch base address */
ldwio r11, 0(r13) /* load slider switches */
stb r11, 0(r12)
br UPDATE_TIME
DO_KEY3:
andi r13, r11, 0b1000 /* check KEY3 */
```

```

beq r13, zero, END_PUSHBUTTON_ISR
bne r17, r0, END_PUSHBUTTON_ISR
movia r12, hours
movia r13, switch_addr /* SW slider switch base address */
ldwio r11, 0(r13) /* load slider switches */
stb r11, 0(r12)
UPDATE_TIME:
movia r10, seconds
stb r0, 0(r10)
movia r11, minutes
movia r12, hours
movia r4, 0
ldb r5, 0(r11)
ldb r6, 0(r12)
call hex_display
br END_PUSHBUTTON_ISR
END_PUSHBUTTON_ISR:
ldw ra, 0(sp) /* Restore all used register to previous */
ldw r10, 4(sp)
ldw r11, 8(sp)
ldw r12, 12(sp)
ldw r13, 16(sp)
ldw r16, 20(sp)
ldw r17, 24(sp)
ldw r4, 28(sp)
ldw r5, 32(sp)
ldw r6, 36(sp)
addi sp, sp, 40
ret
.end
//interval_timer
.include "key_codes.s"
.global INTERVAL_TIMER_ISR
INTERVAL_TIMER_ISR:
subi sp, sp, 48 /* reserve space on the stack */
stw ra, 0(sp)
stw r16, 8(sp)
stw r10, 12(sp)
stw r11, 16(sp)
stw r12, 20(sp)
stw r13, 24(sp)
stw r14, 28(sp)
stw r15, 32(sp)
stw r17, 36(sp)
stw r5, 40(sp)
stw r6, 44(sp)
movia r10, timer_addr /* interval timer base address */
sthio r0, 0(r10) /* clear the interrupt */
movia r10, 59
movia r11, 23
movia r12, 0
movia r13, 0
movia r14, 0
/* получаем значения из памяти */
movia r17, hours
movia r16, minutes
movia r15, seconds
ldb r6, 0(r17)
ldb r5, 0(r16)
ldb r4, 0(r15)
/* проверяем полученные значения */
bne r4, r10, check_minutes
movia r12, 1

```

```

movia r4, 0
check_minutes:
bne r5, r10, check_hours
beq r12, r0, check_hours
movia r13, 1
movia r12, 0
check_hours:
bne r6, r11, conditions
beq r13, r0, conditions
movia r13, 0
conditions:
bne r14, r0, start_day
bne r13, r0, add_hour
bne r12, r0, add_minute
br add_seconds
add_minute:
addi r5, r5, 1
br END_INTERVAL_TIMER_ISR
add_hour:
addi r6, r6, 1
movia r5, 0
br END_INTERVAL_TIMER_ISR
start_day:
movia r5, 0
movia r6, 0
br END_INTERVAL_TIMER_ISR
add_seconds:
addi r4, r4, 1
END_INTERVAL_TIMER_ISR:
/* заносим значения в память */
stb r6, 0(r17)
stb r5, 0(r16)
stb r4, 0(r15)
call hex_display
ldw ra, 0(sp) /* Restore all used register to previous */
ldw r4, 4(sp)
ldw r16, 8(sp)
ldw r10, 12(sp)
ldw r11, 16(sp)
ldw r12, 20(sp)
ldw r13, 24(sp)
ldw r14, 28(sp)
ldw r15, 32(sp)
ldw r17, 36(sp)
ldw r5, 40(sp)
addi sp, sp, 48 /* release the reserved space on the stack */
ret
.data
.global seconds
seconds:
.byte 0
.global minutes
minutes:
.byte 0
.global hours
hours:
.byte 0
.end
//get_dec
.text
.global get_dec_and_edin
get_dec_and_edin:
subi sp, sp, 20

```

```

stw r3, 0(sp)
stw r4, 4(sp)
stw r5, 8(sp)
stw r6, 12(sp)
stw ra, 16(sp)
mov r3, r1 # r1 - Число для отображения
# Разделим на десятки и единицы
movia r4, 10
divu r5, r3, r4 # десятки
mul r6, r5, r4 # r6 = (r3 / r4) * r4
sub r4, r3, r6 # единицы
mov r1, r5 # десятки
mov r2, r4 # единицы
ldw r3, 0(sp)
ldw r4, 4(sp)
ldw r5, 8(sp)
ldw r6, 12(sp)
ldw ra, 16(sp)
ret
.end
//exception_handler
.section .reset, "ax"
movia r2, _start
jmp r2 /* branch to main program */
.section .exceptions, "ax"
.global EXCEPTION_HANDLER
EXCEPTION_HANDLER:
subi sp, sp, 16 /* make room on the stack */
stw et, 0(sp)
rdctl et, ctl4
beq et, r0, SKIP_EA_DEC /* interrupt is not external */
subi ea, ea, 4 /* must decrement ea by one instruction */
/* for external interrupts, so that the */
/* interrupted instruction will be run after eret */
SKIP_EA_DEC:
stw ea, 4(sp) /* save all used registers on the Stack */
stw ra, 8(sp) /* needed if call inst is used */
stw r22, 12(sp)
rdctl et, ctl4
bne et, r0, CHECK_LEVEL_0 /* exception is an external interrupt */
NOT_EI: /* exception must be unimplemented instruction or TRAP */
br END_ISR /* instruction. This code does not handle those cases */
CHECK_LEVEL_0: /* interval timer is interrupt level 0 */
andi r22, et, 0b1
beq r22, r0, CHECK_LEVEL_1
call INTERVAL_TIMER_ISR
br END_ISR
CHECK_LEVEL_1: /* pushbutton port is interrupt level 1 */
andi r22, et, 0b10
beq r22, r0, END_ISR /* other interrupt levels are not handled in this code */
call PUSHBUTTON_ISR
END_ISR:
ldw et, 0(sp) /* restore all used register to previous values */
ldw ea, 4(sp)
ldw ra, 8(sp) /* needed if call inst is used */
ldw r22, 12(sp)
addi sp, sp, 16
eret
.end

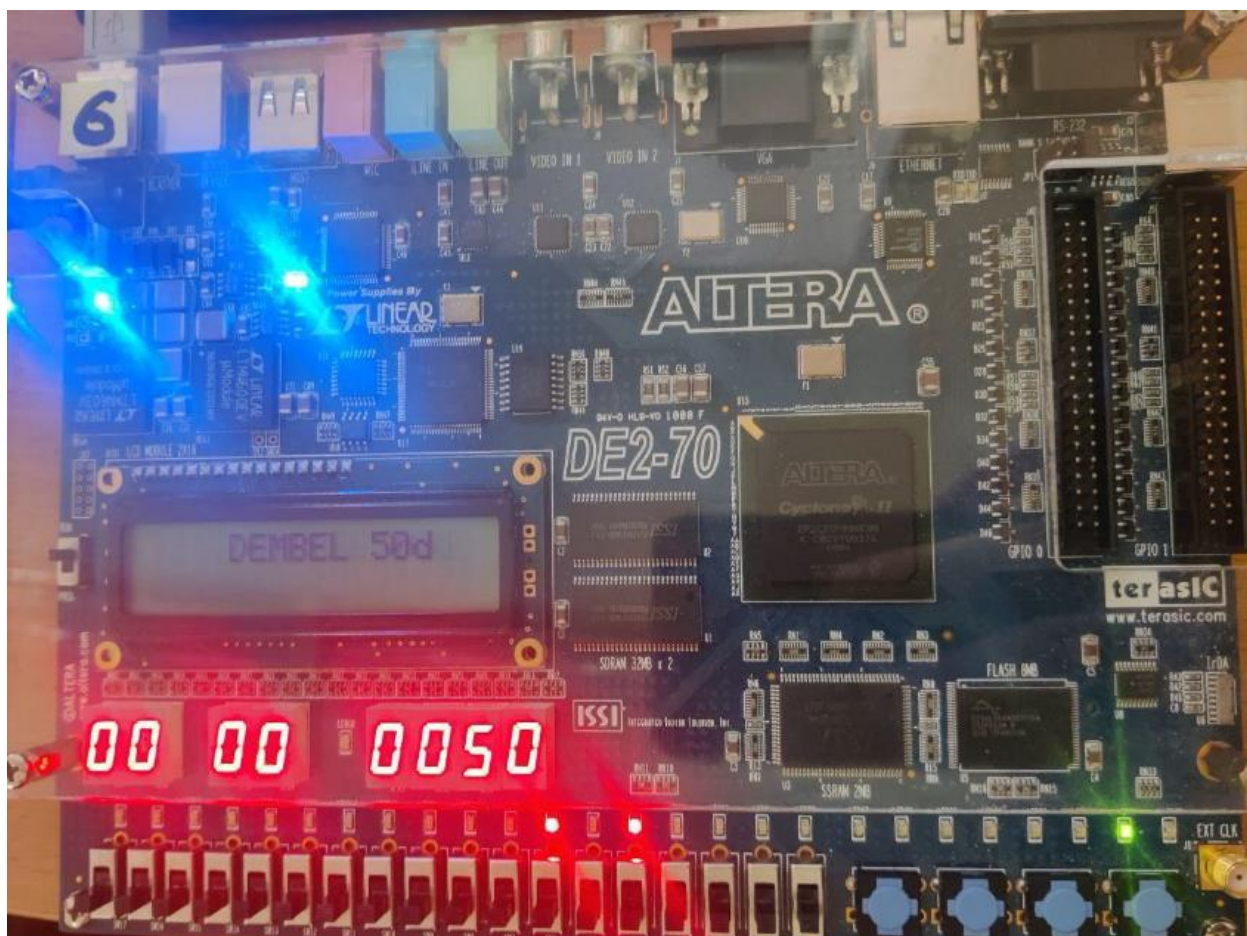
```

Экспериментально определили действия программы при выполнении сброса процессорной системы. Для этого поставим по нулевому адресу в ОП контрольную точку. Затем запустим программу и нажмем кнопку *KEY0*. Далее выполним программу по шагам. При нажатии *KEY0* программа переходит по нулевому адресу в ОП. Далее выполняется переход до метки *\_start*, соответственно выполнение программы начинается заново.

Запрещаем прерывания от отдельных кнопок, изменяя содержимое регистра маски, соответствующего PIO. Необходимо устанавливать биты (0000xxx0) в 0 по адресу 0x10000058 для запрета прерывания какой-либо из кнопок.

Чтобы запретить прерывания от кнопок, необходимо чтобы в *ienable* первый бит был равен 0.

Для запрета прерывания от кнопок через регистр *status* нужно, чтобы бит *pie* был равен 0, а для разрешения – 1.



## Часть 5. Программное прерывание **trap** и невыполнимые команды

1 Вставим в текст программы команду **trap**. Модифицируем обработчик прерываний таким образом, чтобы в случае обнаружения команды **trap**, он передал управление процедуре, которая выведет строку «trap» на LCD экран. Команда **trap** является аналогом команды **call**. Команда **trap** принудительно вызывает обработчик прерывания.



Команда **trap** относится к типу R.  $0x003b683a = 0b\ 00000\ 00000\ 11101\ 10110100000\ 111010$ . Поле C = r29 = ea.

Экспериментально определим как выполняется команда **mul r12,r10,r11**. Перед выполнением команды в регистровом окне АМР задавайте различные значения сомножителей с использованием их знакового представления и наблюдаем результат в регистре r12.

|     |            |     |            |     |            |     |            |
|-----|------------|-----|------------|-----|------------|-----|------------|
| r10 | 0x00000023 | r10 | 0x80000023 | r10 | 0x00000023 | r10 | 0x80000023 |
| r11 | 0x00000002 | r11 | 0x80000002 | r11 | 0x80000002 | r11 | 0x00000002 |
| r12 | 0x00000046 | r12 | 0x80000046 | r12 | 0x00000046 | r12 | 0x00000046 |
| ..  | .....      | ..  | .....      | ..  | .....      | ..  | .....      |

Для выполнения оставшихся пунктов задания следует использовать другую процессорную систему «DE 2-115 Basic Comput-er». В ней используется экономная версия (e) процессора NIOS II, в которой не поддерживается аппаратное выполнение команд умножения и деления. Выполнили команду *Settings > System settings*. В появившемся окне следует указать используемую процессорную систему. Для этого в поле *Select a system* щелкнем мышью по значку раскрытия списка. В появившемся списке выберите строку «DE2-70 Basic Computer» или «DE2-115 Basic Computer».

Выполнили загрузку выбранной процессорной системы в кристалл ПЛИС на стенде, используя команду *Download System* из меню *Actions*.

Вставили в текст программы рассмотренную в пункте 5.2 команду умножения. Экспериментально определим поведение программы при выполнении этой команды. Следует напомнить, что во вновь установленной процессорной системе не поддерживается аппаратное умножение.

Модифицировали обработчик исключений таким образом, чтобы он передал управление процедуре, которая эмулирует выполнение команды **mul**. Результат работы команды **mul** в режиме **basic** совпадает с результатом в режиме **media**, который был приведен ранее.

|            | +0x0     | +0x4     | +0x8     | +0xc     |
|------------|----------|----------|----------|----------|
| 0x00000000 | 00000000 | 00000023 | 00000002 | 00000046 |

|            | +0x0     | +0x4     | +0x8     | +0xc     |
|------------|----------|----------|----------|----------|
| 0x00000000 | 00000000 | 80000023 | 80000002 | 80000046 |

|            | +0x0     | +0x4     | +0x8     | +0xc     |
|------------|----------|----------|----------|----------|
| 0x00000000 | 00000000 | 00000023 | 80000002 | 80000046 |

|            | +0x0     | +0x4     | +0x8     | +0xc     |
|------------|----------|----------|----------|----------|
| 0x00000000 | 00000000 | 00000023 | 00000002 | 00000046 |

## Вывод

В результате выполнения лабораторной работы, были изучены возможные способы ввода/вывода информации в процессорной системе. В лабораторной были затронуты индикаторы LEDR, а также LCD дисплей и была проведена работа по выводу информации на них.