Министерство науки и высшего образования Российской Федерации
Мытищинский филиал
Федеральное государственное автономное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ космический

КАФЕДРА К-3

отчет

ПО ЛАБОРАТОРНОЙ РАБОТЕ № 5

по ДИСЦИПЛИНЕ

«ОРГАНИЗАЦИЯ ЭВМ И СИСТЕМ»

Студент К3-66Б (Группа) Чернов В. Д. (И.О.Фамилия)

Студент К3-66Б (Группа) Братов А. Р. (И.О.Фамилия)

Преподаватель Ефремов Н. В. (И.О.Фамилия)

2025 г.

Цель работы:

Изучить, каким образом выполняется взаимодействие программных компонентов друг с другом; уяснить, как работает стек, как в процессорной системе использовать исходные данные, размещенные в отдельном файле.

Выполнение работы

1.1. - 1.2. Возьмём за основу программу по нахождению наибольшего числа из списка и оформим часть, где осуществлялся поиск максимального числа, в подпрограмму с именем Max. Для вызова использовалась команда call, а для возврата из подпрограммы команда ret.

```
/* Программа выполняет поиск максимального числа в списке целых чисел */
.text                /* секция кода */
.global _start
.equ ledr, 0x10000000
_start:
   movia   r4, RESULT  # В регистр r4 запишем адрес ОП, куда поместим результат
   ldw  r5, 4(r4)       # Считываем в регистр r5 значение N -количество чисел в списке
   addi r6, r4, 8   # Вычисляем адрес памяти, с которого начинают располагаться числа для поиска и записываем его в r6

 call Max

   stw  r7, (r4)        # Записываем найденное число в ячейку памяти RESULT
   movia r9, ledr
   stwio r7, (r9)
STOP:
   br   STOP           # Бесконечный цикл. Завершаем программу


/* Далее размещается область данных программы */
RESULT:
.skip   4       # Резервируем 4 байта для записи результата
N:
.word 14         # Количество чисел в списке
NUMBERS:
.word   4, 5, 3, 6, 1, 8, 2, -7, -3, 2, -2, 0x00030783, 0, 14     # Числа из списка

Max:
```

```
    ldw  r7, (r6)       # В регистр r7 из ОП считывам первое число из списка
LOOP:
    subi r5, r5, 1   # Уменьшаем значение количества чисел в списке
    beq  r5, r0, DONE    # Если значение регистра r5 равно 0, то выходим из цикла
    addi r6, r6, 4   # Увеличиваем адрес памяти на 4 для перехода к следующему числу в списке
    ldw  r8, (r6)       # Считываем в r8 следующее число из списка
    bge  r7, r8, LOOP    # Если найденное максимальное число больше или равно считанному, то
возвращаемся в начало цикла
    mov  r7, r8  # Иначе, обновляем в r7 максимальное число
    br   LOOP           # Безусловный переход в начало цикла

DONE:
    ret

.end
```

Листинг 1

Как мы видим, программа правильно выполняет свою работу, светодиоды тоже горят.

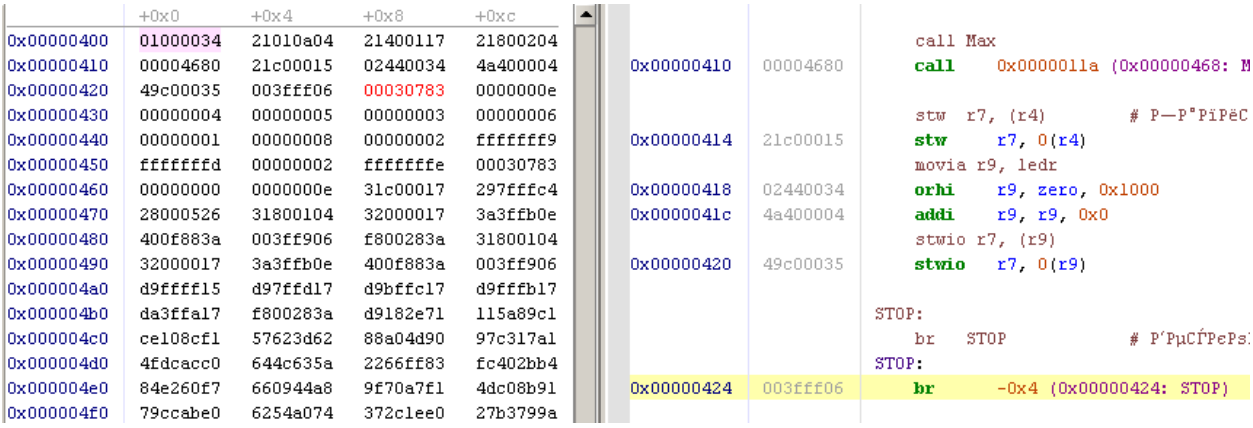


Рисунок 1

1.3. Поместим список чисел в отдельный сегмент данных в начале статической памяти.

```
/* Программа выполняет поиск максимального числа в списке целых чисел */
.text               /* секция кода */
.global _start
.equ ledr, 0x10000000
_start:
    movia   r4, RESULT  # В регистр r4 запишем адрес ОП, куда поместим результат
    ldw  r5, 4(r4)       # Считываем в регистр r5 значение N -количество чисел в списке
    addi r6, r4, 8   # Вычисляем адрес памяти, с которого начинают располагаться числа для поиска и
записываем его в r6
```

```
    call Max

    stw  r7, (r4)       # Записываем найденное число в ячейку памяти RESULT
    movia r9, ledr
    stwio r7, (r9)
STOP:
    br   STOP           # Бесконечный цикл. Завершаем программу


Max:
    ldw  r7, (r6)       # В регистр r7 из ОП считывам первое число из списка
LOOP:
    subi r5, r5, 1   # Уменьшаем значение количества чисел в списке
    beq  r5, r0, DONE    # Если значение регистра r5 равно 0, то выходим из цикла
    addi r6, r6, 4   # Увеличиваем адрес памяти на 4 для перехода к следующему числу в списке
    ldw  r8, (r6)       # Считываем в r8 следующее число из списка
     bge   r7, r8, LOOP     # Если найденное максимальное число больше или равно считанному, то возвращаемся в начало цикла
    mov  r7, r8  # Иначе, обновляем в r7 максимальное число
    br   LOOP           # Безусловный переход в начало цикла

DONE:
    ret


.data
/* Далее размещается область данных программы */
RESULT:
.skip   4       # Резервируем 4 байта для записи результата
N:
.word 14        # Количество чисел в списке
NUMBERS:
.word   4, 5, 3, 6, 1, 8, 2, -7, -3, 2, -2, 0x00030783, 0, 14     # Числа из списка
.end
```

Листинг 2

Как мы видим, данные по-прежнему отображаются в памяти и регистрах корректно.

	+0x0	+0x4	+0x8	+0xc
0x08000000	00030783	0000000e	00000004	00000005
0x08000010	00000003	00000006	00000001	00000008
0x08000020	00000002	fffffff9	fffffffd	00000002
0x08000030	fffffffe	00030783	00000000	0000000e

Рисунок 2

r7	0x00030783

Рисунок 3

1.4. Заменим команду call на callr. Данная команда вызывает подпрограмму по адресу, который находится в регистре.

```
movia r1, Max
callr r1
```

Листинг 3

1.5. Выполним декодирование машинных кодов команд call, callr и ret.

```
                call Max
0x00000410  00004280    call    0x0000010a (0x00000428: Max)
```

Рисунок 4

```
0x00000418  083ee83a    callr   r1
```

Рисунок 5

```
0x00000448  f800283a    ret
```

Рисунок 6

Рассмотрим команду call, которая принадлежит к типу J.

0x00004280 = 0b 00000000000000000100001010 000000

IMMED26 = 0b00000000000000000100001010 = 0x10a

Дополненное поле IMMED26 0b0000000000000000000010000101000 = 0x428 (адрес подпрограммы Max)

Рассмотрим команду callr, которая принадлежит к типу R.

0x83ee8ea = 0b 00001 00000 11111 01110100000 111010

Поле A = 1, что означает r1. Поле B = 0, что означает r0. Поле C = 31, что означает r31, который является регистром, где хранится адрес возврата.

Рассмотрим команду ret, которая относится к типу R.

0xf800283a = 0b 11111 00000 00000 00010100000 111010

Поле A = 31, что означает r31, который является регистром, где хранится адрес возврата. Поле B = 0, что означает r0. Поле C = 0, что означает r0.

1.6. Оформим вывод результата на красные светодиоды в виде самостоятельной процедуры Out_on_LEDR.

```
/* Программа выполняет поиск максимального числа в списке целых чисел */
.text                /* секция кода */
.global _start
.equ ledr, 0x10000000
_start:
    movia   r4, RESULT  # В регистр r4 запишем адрес ОП, куда поместим результат
    ldw  r5, 4(r4)       # Считываем в регистр r5 значение N -количество чисел в списке
   addi r6, r4, 8   # Вычисляем адрес памяти, с которого начинают располагаться числа для поиска и записываем его в r6

  call Max

    stw  r7, (r4)        # Записываем найденное число в ячейку памяти RESULT

  call Out_on_LEDR

STOP:
    br   STOP           # Бесконечный цикл. Завершаем программу

Out_on_LEDR:
  movia r9, ledr
    stwio r7, (r9)
  ret

Max:
  ldw  r7, (r6)        # В регистр r7 из ОП считывам первое число из списка
LOOP:
    subi r5, r5, 1   # Уменьшаем значение количества чисел в списке
    beq  r5, r0, DONE    # Если значение регистра r5 равно 0, то выходим из цикла
    addi r6, r6, 4   # Увеличиваем адрес памяти на 4 для перехода к следующему числу в списке
    ldw  r8, (r6)        # Считываем в r8 следующее число из списка
```

```
   bge  r7, r8, LOOP    # Если найденное максимальное число больше или равно считанному, то возвращаемся в начало цикла
    mov  r7, r8  # Иначе, обновляем в r7 максимальное число
    br   LOOP           # Безусловный переход в начало цикла

DONE:
  ret


.data
/* Далее размещается область данных программы */
RESULT:
.skip   4       # Резервируем 4 байта для записи результата
N:
.word 14        # Количество чисел в списке
NUMBERS:
.word   4, 5, 3, 6, 1, 8, 2, -7, -3, 2, -2, 0x00030783, 0, 14     # Числа из списка
.end
```

Листинг 4

1.7. Схема взаимодействия основной программы с процедурами. В основной программе вызывается подпрограмма Max, которая находит наибольшее число, затем происходит возврат в основную программу, далее вызывается подпрограмма Out_on_LEDR, отображающая число на светодиодах, а затем происходит возврат в основную программу.

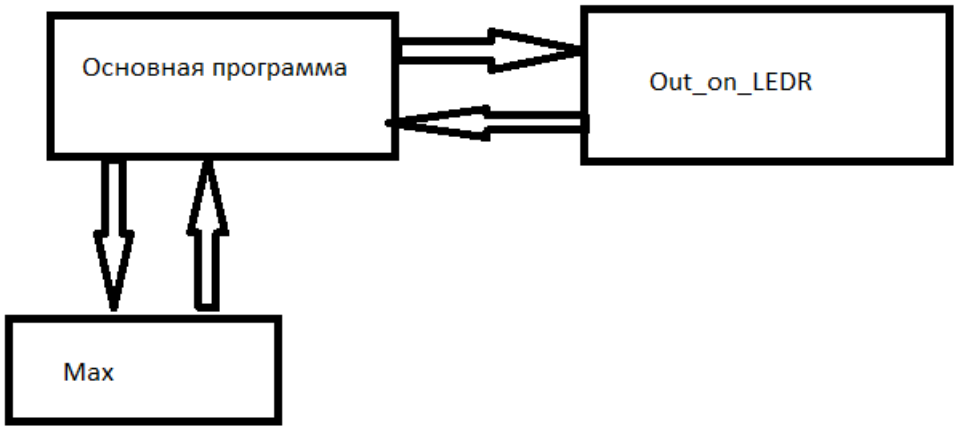


Рисунок 7

2.1. Модифицируем программу из предыдущей части следующим образом: будем вызвать процедуру Out_on_LEDR внутри процедуры Max.

```
Out_on_LEDR:
 movia r9, ledr
   stwio r7, (r9)
 ret

Max:
 ldw  r7, (r6)       # В регистр r7 из ОП считывам первое число из списка
LOOP:
   subi r5, r5, 1   # Уменьшаем значение количества чисел в списке
   beq  r5, r0, DONE    # Если значение регистра r5 равно 0, то выходим из цикла
   addi r6, r6, 4   # Увеличиваем адрес памяти на 4 для перехода к следующему числу в списке
   ldw  r8, (r6)       # Считываем в r8 следующее число из списка
  bge  r7, r8, LOOP    # Если найденное максимальное число больше или равно считанному, то возвращаемся в начало цикла
   mov  r7, r8  # Иначе, обновляем в r7 максимальное число
   br   LOOP           # Безусловный переход в начало цикла

DONE:
 call Out_on_LEDR
 ret
```

Листинг 5

2.2. Программа не осуществляет возврат в основную программу из-за того, что адрес возврата является адресом внутри процедуры Max. То есть сложилась такая ситуация: в основной программе вызвали процедуру Max, адрес возврата указывает на следующую команду в основной программе, затем в процедуре Max вызывается подпрограмма Out_on_LEDR, при этом адрес возврата меняется на адрес следующей после вызова данной подпрограммы команды, а команда ret в процедуре Max будет возвращать в процедуру Max на ту же самую команду ret.

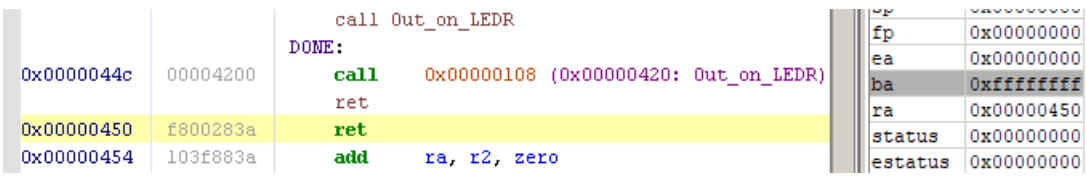


Рисунок 8

Для возврата в основную программу необходимо перед вызовом процедуры Out_on_LEDR сохранить адрес возврата, а после присвоить ему старое значение.

```
Out_on_LEDR:
  movia r9, ledr
    stwio r7, (r9)
  ret

Max:
  ldw  r7, (r6)        # В регистр r7 из ОП считывам первое число из списка
  mov r2, ra     # сохраняем адрес возврата
LOOP:
    subi r5, r5, 1   # Уменьшаем значение количества чисел в списке
    beq  r5, r0, DONE    # Если значение регистра r5 равно 0, то выходим из цикла
    addi r6, r6, 4   # Увеличиваем адрес памяти на 4 для перехода к следующему числу в списке
    ldw  r8, (r6)        # Считываем в r8 следующее число из списка
   bge  r7, r8, LOOP    # Если найденное максимальное число больше или равно считанному, то возвращаемся в начало цикла
    mov  r7, r8  # Иначе, обновляем в r7 максимальное число
    br   LOOP            # Безусловный переход в начало цикла

DONE:
  call Out_on_LEDR
  mov ra, r2    # присваиваем адрес возврата для возврата в основную программу
  ret
```

Листинг 6

2.3. Схема взаимодействия основной программы и процедур.

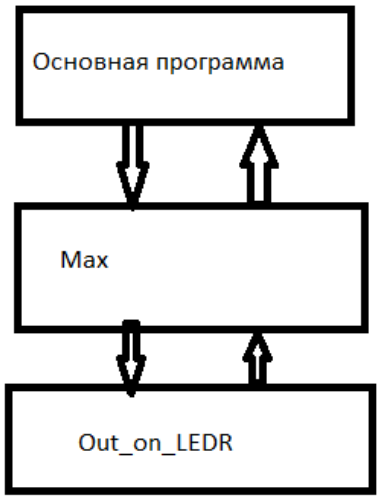


Рисунок 9

3.1. Модифицируем программу из первой части таким образом, чтобы передача параметров процедуре Max осуществлялась через стек. Для этого в основной программе определяем в качестве вершины стека адрес последнего слова используемой в процессорной системе динамической памяти. По нулевому смещению адреса стека будем класть количество чисел, по -8 смещению будем класть адрес первого числа из списка. В начале подпрограммы Max будем читать из стека переданные значения, а в конце по -4 смещению заносить максимальное число.

```
/* Программа выполняет поиск максимального числа в списке целых чисел */
.text                /* секция кода */
.global _start
.equ ledr, 0x10000000
_start:
    movia   r4, RESULT  # В регистр r4 запишем адрес ОП, куда поместим результат
    ldw  r5, 4(r4)       # Считываем в регистр r5 значение N -количество чисел в списке
   addi r6, r4, 8   # Вычисляем адрес памяти, с которого начинают располагаться числа для поиска и записываем
его в r6

  movia sp, 0x07FFFFFC # адрес последнего слова в динамической памяти
  stw r5, 0(sp)        # количество чисел в списке
  stw r6, -8(sp)       # адрес первого числа из списка

  call Max

  stw r7, (r4)         # Записываем найденное число в ячейку памяти RESULT

  call Out_on_LEDR

STOP:
    br   STOP           # Бесконечный цикл. Завершаем программу

Out_on_LEDR:
  movia r9, ledr
    stwio r7, (r9)
  ret

Max:
    ldw r5, 0(sp)
```

```
    ldw r6, -8(sp)
    ldw  r7, (r6)        # В регистр r7 из ОП считывам первое число из списка
LOOP:
    subi r5, r5, 1   # Уменьшаем значение количества чисел в списке
    beq  r5, r0, DONE    # Если значение регистра r5 равно 0, то выходим из цикла
    addi r6, r6, 4   # Увеличиваем адрес памяти на 4 для перехода к следующему числу в списке
    ldw  r8, (r6)        # Считываем в r8 следующее число из списка
   bge  r7, r8, LOOP    # Если найденное максимальное число больше или равно считанному, то возвращаемся в
начало цикла
    mov  r7, r8  # Иначе, обновляем в r7 максимальное число
    br   LOOP            # Безусловный переход в начало цикла

DONE:
    stw r7, -4(sp)
  ret


.data
/* Далее размещается область данных программы */
RESULT:
.skip   4       # Резервируем 4 байта для записи результата
N:
.word 14        # Количество чисел в списке
NUMBERS:
.word   4, 5, 3, 6, 1, 8, 2, -7, -3, 2, -2, 0x00030783, 0, 14     # Числа из списка
.end
```

Листинг 7

Посмотрим содержимое стека после выполнения программы. Как и ожидалось, по -8 смещению находится адрес первого числа из списка, далее найденное число, а затем количество чисел.

Рисунок 10

3.2. Воспользуемся общепринятым приёмом – сохранение содержимого регистров, используемых в подпрограмме, в стеке в начале работы подпрограммы и восстановление их перед завершением подпрограммы.

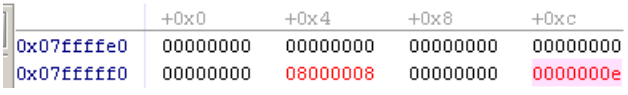
```
/* Программа выполняет поиск максимального числа в списке целых чисел */
.text               /* секция кода */
.global _start
```

```
.equ ledr, 0x10000000
_start:
    movia   r4, RESULT  # В регистр r4 запишем адрес ОП, куда поместим результат
    ldw  r5, 4(r4)       # Считываем в регистр r5 значение N -количество чисел в списке
   addi r6, r4, 8   # Вычисляем адрес памяти, с которого начинают располагаться числа для поиска и записываем
его в r6

  movia sp, 0x07FFFFC # адрес последнего слова в динамической памяти
  stw r5, 0(sp)        # количество чисел в списке
  stw r6, -8(sp)       # адрес первого числа из списка

  movia r5, 0x555
  movia r6, 0x666
  movia r7, 0x777
  movia r8, 0x888
  call Max

  ldw r10, -4(sp)       # Записываем найденное число в ячейку памяти RESULT
  stw r10, (r4)

  call Out_on_LEDR



STOP:
    br   STOP           # Бесконечный цикл. Завершаем программу

Out_on_LEDR:
  movia r9, ledr
    stwio r10, (r9)
  ret

Max:
  stw r5, -12(sp)
  stw r6, -16(sp)
  stw r7, -20(sp)
  stw r8, -24(sp)

    ldw r5, 0(sp)
    ldw r6, -8(sp)

  ldw  r7, (r6)        # В регистр r7 из ОП считывам первое число из списка
LOOP:
```

```
    subi r5, r5, 1   # Уменьшаем значение количества чисел в списке
    beq  r5, r0, DONE    # Если значение регистра r5 равно 0, то выходим из цикла
    addi r6, r6, 4   # Увеличиваем адрес памяти на 4 для перехода к следующему числу в списке
    ldw  r8, (r6)        # Считываем в r8 следующее число из списка
   bge  r7, r8, LOOP    # Если найденное максимальное число больше или равно считанному, то возвращаемся в
начало цикла
    mov  r7, r8  # Иначе, обновляем в r7 максимальное число
    br   LOOP            # Безусловный переход в начало цикла

DONE:
    stw r7, -4(sp)

  ldw r5, -12(sp)
  ldw r6, -16(sp)
  ldw r7, -20(sp)
  ldw r8, -24(sp)

  ret


.data
/* Далее размещается область данных программы */
RESULT:
.skip   4       # Резервируем 4 байта для записи результата
N:
.word 14        # Количество чисел в списке
NUMBERS:
.word   4, 5, 3, 6, 1, 8, 2, -7, -3, 2, -2, 0x00030783, 0, 14     # Числа из списка
.end
```

Листинг 8

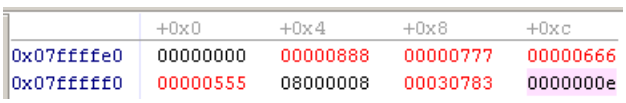
Стек перед использованием команды call.



	+0x0	+0x4	+0x8	+0xc
0x07fffe0	00000000	00000000	00000000	00000000
0x07ffff0	00000000	08000008	00000000	0000000e

Рисунок 11

Стек после выполнения команды ret.



	+0x0	+0x4	+0x8	+0xc
0x07fffe0	00000000	00000888	00000777	00000666
0x07ffff0	00000555	08000008	00030783	0000000e

Рисунок 12

Значения регистров после выполнения программы.

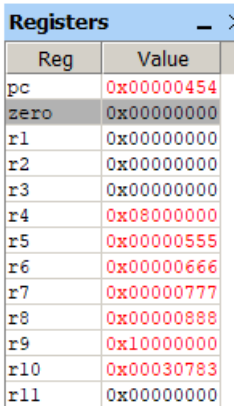


Рисунок 13

4.1. Создадим файл и поместим в него список чисел. Для этого используем шестнадцатеричную форму представления чисел. В качестве разделителя может использоваться любой символ. Загрузим файл в оперативную память, начиная с заданного адреса. Используем команду *Load file into memory* из меню Action приложения IMP. В появившемся окне зададим имя файла с указанием пути к нему, в поле «формат файла» окна браузера зададим *Delimited hexadecimal value file*, символ разделителя и начальный адрес загрузки в оперативной памяти. Список чисел:

4,5,3,6,1,8,2,fffffff9,fffffffd,2,fffffffe,00030783,0,e.

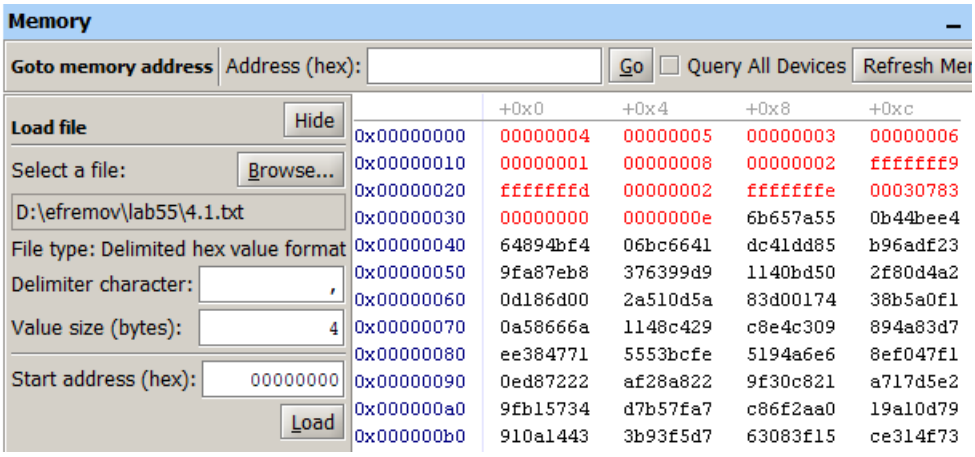


Рисунок 14

4.2. Загрузим файл в статическую память.

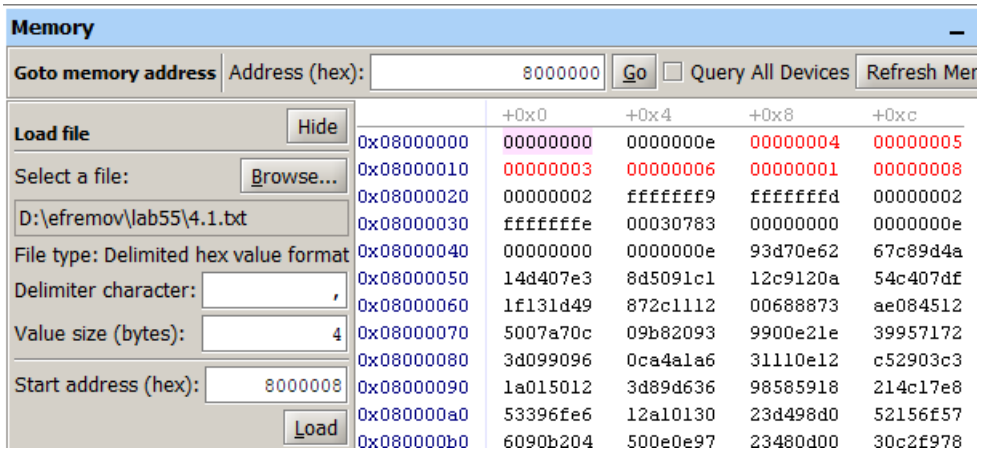


Рисунок 15

4.3. Модифицируем программу из первой части таким образом, чтобы она передавала подпрограмме через регистры число слов в списке, соответствующее подготовленному файлу и начальный адрес ОП, куда был загружен файл. После получения результата через регистр, основная программа выводит его на светодиоды.

```
/* Программа выполняет поиск максимального числа в списке целых чисел */
.text                /* секция кода */
.global _start
.equ ledr, 0x10000000
_start:
    movia   r4, RESULT  # В регистр r4 запишем адрес ОП, куда поместим результат
    ldw  r5, 4(r4)       # Считываем в регистр r5 значение N -количество чисел в списке
    movia   r6, 0x8000050

  call Max

    stw  r7, (r4)        # Записываем найденное число в ячейку памяти RESULT

  call Out_on_LEDR

STOP:
    br   STOP            # Бесконечный цикл. Завершаем программу

Out_on_LEDR:
  movia r9, ledr
    stwio r7, (r9)
  ret
```

```
Max:
    ldw  r7, (r6)        # В регистр r7 из ОП считывам первое число из списка
LOOP:
    subi r5, r5, 1    # Уменьшаем значение количества чисел в списке
    beq  r5, r0, DONE     # Если значение регистра r5 равно 0, то выходим из цикла
    addi r6, r6, 4    # Увеличиваем адрес памяти на 4 для перехода к следующему числу в списке
    ldw  r8, (r6)        # Считываем в r8 следующее число из списка
   bge  r7, r8, LOOP    # Если найденное максимальное число больше или равно считанному, то возвращаемся в начало цикла
    mov  r7, r8  # Иначе, обновляем в r7 максимальное число
    br   LOOP            # Безусловный переход в начало цикла

DONE:
  ret


.data
/* Далее размещается область данных программы */
RESULT:
.skip   4       # Резервируем 4 байта для записи результата
N:
.word 14         # Количество чисел в списке
.end
```

Листинг 9

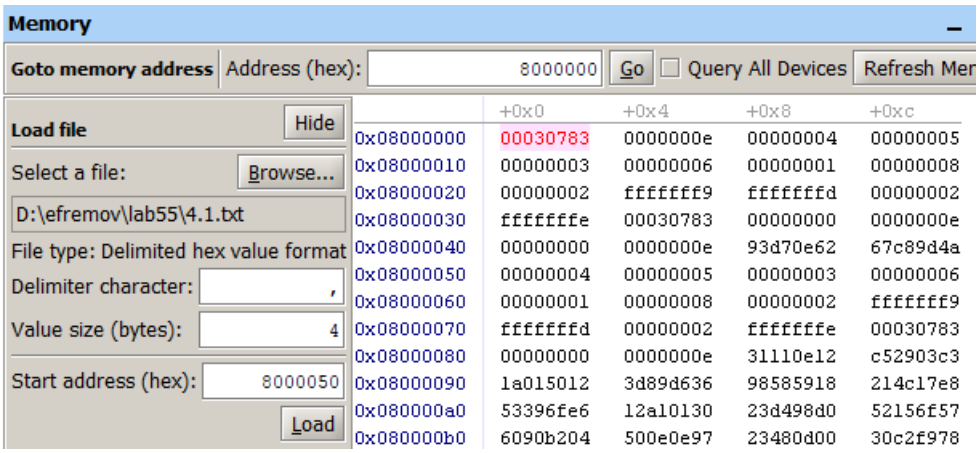


Рисунок 16

5.1. Напишем процедуру Copy, которая выполняет копирование одной области ОП в другую область памяти, начиная с заданного адреса. Используем

следующие параметры: начальный и конечный адреса копируемой области, адрес в ОП, с которого следует разместить копируемый фрагмент памяти.

```
# r13 - начало
# r14 - конец
# r15 - адрес записи
LOOP1:
  ldb r16, (r13)
  stbu r16, (r15)
  addi r15, r15, 1
  addi r13, r13, 1
  bge r14, r13, LOOP1
  ret
```

Листинг 10

5.2. Напишем фрагмент программы, которая будет выполнять копирование части сегмента кода, включая основную программу и две процедуры *Max* и *Out_on_Ledr* (с метки start_ до метки *Copy* -4) по адресу в ОП, смещенному на 0x1000 от начала сегмента кода. Поставим метку на первой команде созданного фрагмента кода. Добавим фрагмент к файлу с функцией *Copy*.

```
.global COPY
COPY:
  movia r13, _start
  movia r14, COPY-4
  movia r15, 0x1000

# r13 - начало
# r14 - конец
# r15 - адрес записи
LOOP1:
  ldbu r16, (r13)
  stb r16, (r15)
  addi r15, r15, 1
  addi r13, r13, 1
  bge r14, r13, LOOP1
  ret
.end
```

Листинг 11

5.3. Выполняем компиляцию и загрузку обновленной программы. Мы можем наблюдать новый начальный адрес. (Данная строка горела жёлтым, фото было сделано на выполнении другой команды).

```
                                .global COPY
                                COPY:
                                    movia r13, _start
                                COPY:
0x000004b4  03400034            orhi    r13, zero, 0x0
```

Рисунок 17

5.4. Поставим контрольную точку в конце фрагмента и выполним его в автоматическом режиме.

```
                                .global COPY
                                COPY:
                                    movia r13, _start
                                COPY:
0x000004b4  03400034            orhi    r13, zero, 0x0
0x000004b8  6b410004            addi    r13, r13, 0x400
                                movia r14, COPY-4
0x000004bc  03800034            orhi    r14, zero, 0x0
0x000004c0  73812c04            addi    r14, r14, 0x4b0
                                movia r15, 0x1000
0x000004c4  03c00034            orhi    r15, zero, 0x0
0x000004c8  7bc40004            addi    r15, r15, 0x1000


                                # r13 - PSP°C‡P°P»Ps
                                # r14 - PePsPSPµC†
                                # r15 - PeCńPrP°
                                LOOP1:
                                    ldbu r16, (r13)
                                LOOP1:
0x000004cc  6c000003            ldbu    r16, 0(r13)
                                stb r16, (r15)
0x000004d0  7c000005            stb     r16, 0(r15)
                                addi r15, r15, 1
0x000004d4  7bc00044            addi    r15, r15, 0x1
                                addi r13, r13, 1
0x000004d8  6b400044            addi    r13, r13, 0x1
                                bge r14, r13, LOOP1
0x000004dc  737ffb0e            bge     r14, r13, -0x14 (0x000004cc: LOOP1)
                                ret
0x000004e0  f800283a            ret
```

Рисунок 18

5.5. Убеждаемся, что копирование программы было произведено правильно.

Машинные коды исходной программы:

	+0x0	+0x4	+0x8	+0xc
0x00000400	01020034	21000004	21400117	01820034
0x00000410	31801404	00004340	21c00015	00004240
0x00000420	003fff06	02440034	4a400004	49c00035
0x00000430	f800283a	31c00017	297fffc4	28000526
0x00000440	31800104	32000017	3a3ffb0e	400f883a
0x00000450	003ff906	f800283a	03400034	6b410004
0x00000460	03800034	73811504	03c00034	7bc40004
0x00000470	6c000003	7c000005	7bc00044	6b400044
0x00000480	737ffb0e	f800283a	f800283a	31c00017
0x00000490	297fffc4	28000526	31800104	32000017
0x000004a0	3a3ffb0e	400f883a	003ff906	f800283a

Рисунок 19

Машинные коды скопированной программы:

	+0x0	+0x4	+0x8	+0xc
0x00001000	01020034	21000004	21400117	01820034
0x00001010	31801404	00004340	21c00015	00004240
0x00001020	003fff06	02440034	4a400004	49c00035
0x00001030	f800283a	31c00017	297fffc4	28000526
0x00001040	31800104	32000017	3a3ffb0e	400f883a
0x00001050	003ff906	0df00e3a	0ff08bd1	07f18ffc
0x00001060	0ff02ff2	0ff01f70	0ff01df0	0ff02bf0
0x00001070	4fb01534	0ef00ff9	0ff02fe2	6ff12fe8
0x00001080	0ff80fe0	4ff007e0	2fb18ea2	0ff40372
0x00001090	0fd00ff0	0ee40ff0	0bf00fb0	0ffc2ff6
0x000010a0	0fd00fa2	07f04e60	0ff02350	0fd009f0

Рисунок 20

5.6. Пробуем выполнить скопированный код основной программы, предварительно установив в счетчике команд его адрес.

Основная программа, скопированная на новый адрес.

```
0x00001000  01020034      orhi    r4, zero, 0x800
0x00001004  21000004      addi    r4, r4, 0x0
0x00001008  21400117      ldw     r5, 4(r4)
0x0000100c  21800204      addi    r6, r4, 0x8
0x00001010  06c20034      orhi    sp, zero, 0x800
0x00001014  defffe04      addi    sp, sp, -0x4
0x00001018  d9400015      stw     r5, 0(sp)
0x0000101c  d9bffe15      stw     r6, -8(sp)
0x00001020  01400034      orhi    r5, zero, 0x0
0x00001024  29415544      addi    r5, r5, 0x555
0x00001028  01800034      orhi    r6, zero, 0x0
0x0000102c  31819984      addi    r6, r6, 0x666
0x00001030  01c00034      orhi    r7, zero, 0x0
0x00001034  39cldde4      addi    r7, r7, 0x777
0x00001038  02000034      orhi    r8, zero, 0x0
0x0000103c  42022204      addi    r8, r8, 0x888
0x00001040  00004640      call    0x00000119 (0x00000464: Max)
0x00001044  dabfff17      ldw     r10, -4(sp)
0x00001048  22800015      stw     r10, 0(r4)
0x0000104c  00004540      call    0x00000115 (0x00000454: Out_on_LEDR)
0x00001050  003fff06      br      -0x4 (0x00001050)
```

Рисунок 21

Процедуры, скопированные на новый адрес. При вызове процедур программа обращалась по их исходному адресу, по скопированным процедурам программа не проходила. Это связано с тем, что копируется машинный код, в котором уже прописаны адреса переходов, которые при копировании без постороннего вмешательства не меняются.

```
0x00001054  02440034    orhi    r9, zero, 0x1000
0x00001058  4a400004    addi    r9, r9, 0x0
0x0000105c  4a800035    stwio   r10, 0(r9)
0x00001060  f800283a    ret
0x00001064  d97ffd15    stw     r5, -12(sp)
0x00001068  d9bffc15    stw     r6, -16(sp)
0x0000106c  d9fffb15    stw     r7, -20(sp)
0x00001070  da3ffa15    stw     r8, -24(sp)
0x00001074  d9400017    ldw     r5, 0(sp)
0x00001078  d9bffe17    ldw     r6, -8(sp)
0x0000107c  31c00017    ldw     r7, 0(r6)
0x00001080  297fffc4    addi    r5, r5, -0x1
0x00001084  28000526    beq     r5, zero, 0x14 (0x0000109c)
0x00001088  31800104    addi    r6, r6, 0x4
0x0000108c  32000017    ldw     r8, 0(r6)
0x00001090  3a3ffb0e    bge     r7, r8, -0x14 (0x00001080)
0x00001094  400f883a    add     r7, r8, zero
0x00001098  003ff906    br      -0x1c (0x00001080)
0x0000109c  d9ffff15    stw     r7, -4(sp)
0x000010a0  d97ffd17    ldw     r5, -12(sp)
0x000010a4  d9bffc17    ldw     r6, -16(sp)
0x000010a8  d9fffb17    ldw     r7, -20(sp)
0x000010ac  da3ffa17    ldw     r8, -24(sp)
```

Рисунок 22

Заключение

В данной лабораторной работе я продолжил знакомство с процессорной системой «DE2-115 Media Computer» и с программой Altera Monitor Program. В лабораторной работе была рассмотрена возможность использования подпрограмм, были рассмотрены команды call, callr, ret. Была выполнена передача параметров в подпрограмму с помощью регистров и стека. Также я попробовал осуществить заполнение памяти данными из файла. Была создана программа копирования участка памяти.