Тестирование команд и написание простого алгоритма

Информация

<https://proglib.io/p/7-sposobov-sortirovki-massivov-na-primere-s-s-illyustraciyami-2022-04-20>

<https://www.radiokot.ru/start/mcu_fpga/avr/14/>

<http://easyelectronics.ru/skazhu-paru-slov-o-optimizacii-koda.html>

<https://easyelectronics.ru/avr-uchebnyj-kurs-makroassembler.html>

Для ознакомления с режимом отладки создадим новый проект и напишем следующий шаблон.

reset:

rjmp main

main:

loop:

rjmp loop

Следующий пример должен работать с любым устройством ATtiny или ATmega, имеющим как минимум регистра порта B. Он должен быть запущен с активным представлением ввода-вывода и настроен для вывод. Эта программа настраивает PORTB в качестве вывода, а затем выполняет цикл, считывая все, что присутствует на PINB, увеличивает его на единицу и выводит в PORTB.

reset:

rjmp main

main:

ldi r16, 0xff

out DDRB, r16

loop:

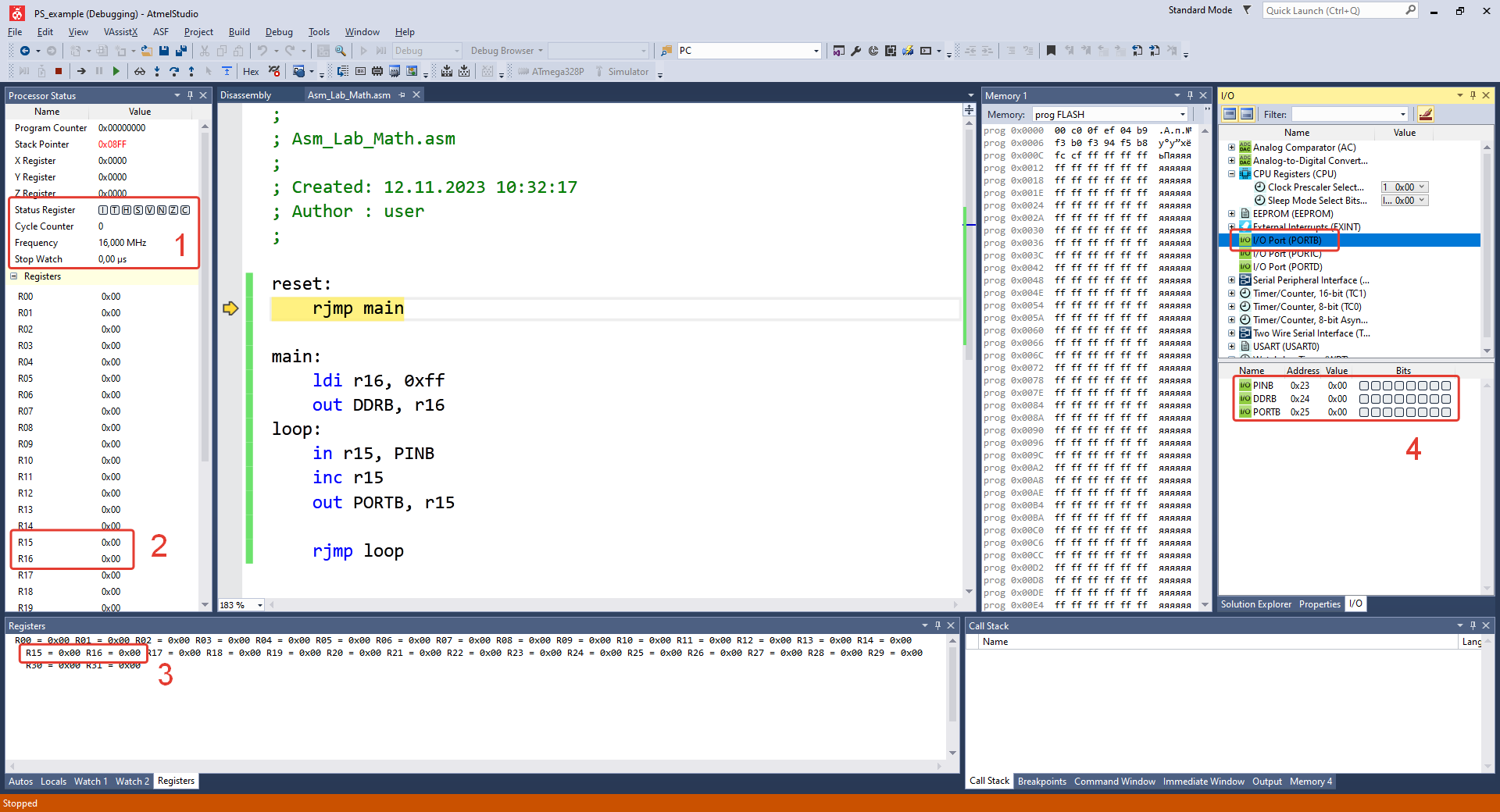
in r15, PINB

inc r15

out PORTB, r15

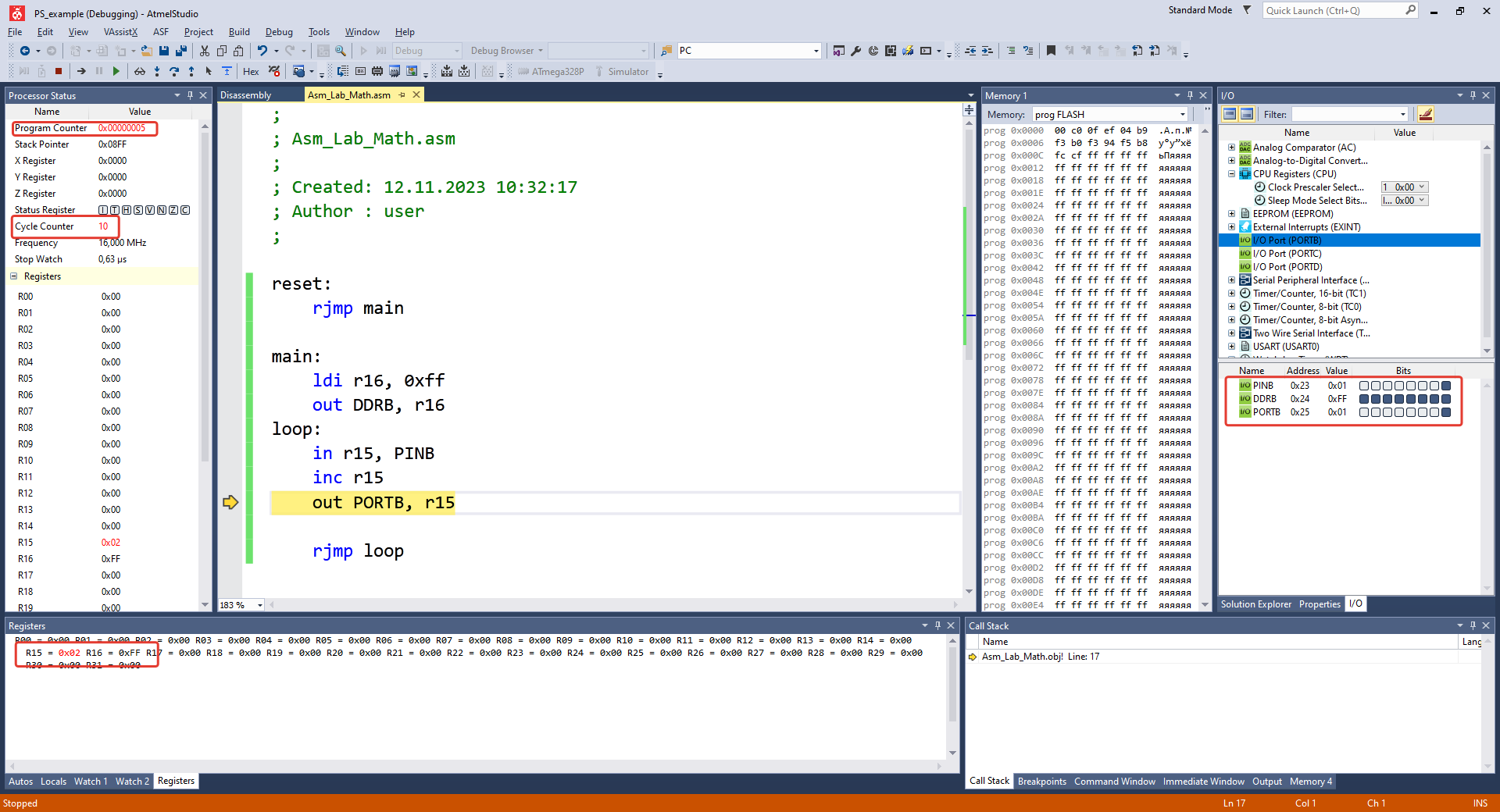
rjmp loop

Для перехода в режим отладки нажимаем Debug->Start Debugging and Break. При этом программа запустится и остановится на первой строке.

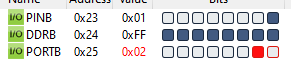


На вкладке Processor Status можно отслеживать Количество пройденных циклов, состояние Status Register (1), а также состояние регистров общего назначения (2). На вкладке Registers также отображается состояние регистров общего назначения (3). На вкладке IO можно отследить состояние регистров ввода/вывода (4).

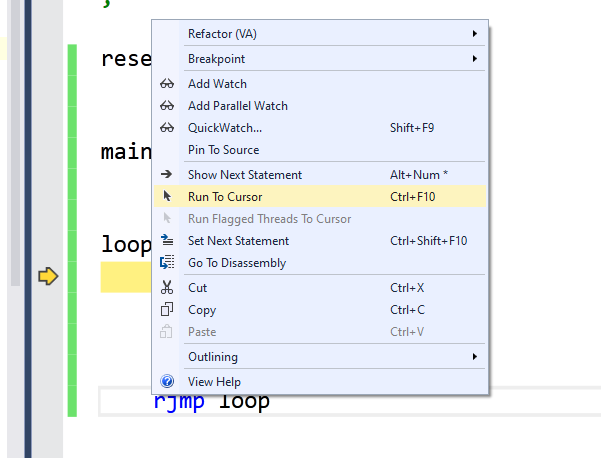
Для перехода на следующую команду следует нажать клавишу F10. Через несколько нажатий получится следующая картина.



При изменении значения поля, оно подсвечивается красным цветом. На следующем шаге значение r15 будет передано в регистр PORTB:



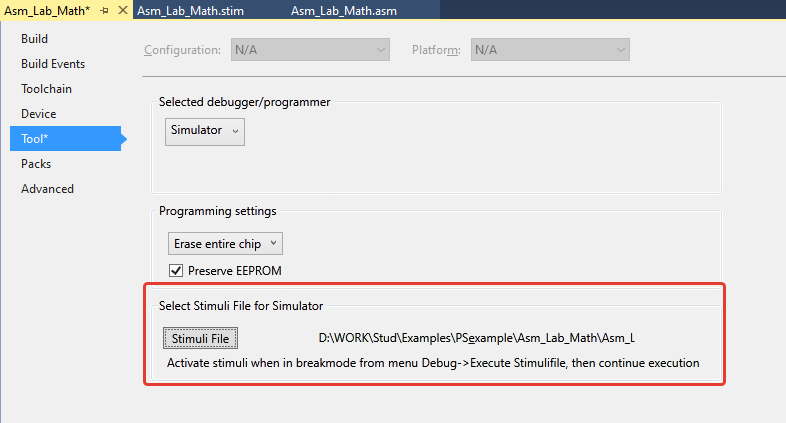
Однако для большого объема кода, особенно с задержкой, нажимать F10 будет неудобно (например, 4800000 раз). В этом случае можно поставить указатель на требуемую строку и выбрать в меню Run to cursor



В этом случае перейдем сразу на строку rjmp loop.

Альтернативным способом отследить состояния портов является использования файла стимуляции. Для данного примера нам нужны регистры: PINB, PORTB.

Для этого добавим в проект новый файл с именем Asm\_Lab\_Math.stim. Также этот файл нужно указать в настройках проекта.



Файл .stim

$log PORTB

$log PINB

$startlog Asm\_Lab\_Math\_log\_output.stim

#102

$endrep

$stoplog

$break

В данном случае выполняется логирование регистров PINB, PORTB, и запись их значений, при изменении, в выходной файл.

Следует отметить, что лог можно сделать только для регистров ввода-вывода.

Запуск стимуляции выполняется в три шага:

1. Debug->Start Debugging and Break
2. Debug->Execute Stimulifile
3. Debug->Continue

После этого можно заметить, что счетчик циклов равен 103.

В файле Asm\_Lab\_Math\_log\_output.stim будет следующая запись (для удобства отображена в 4четыре столбца):

#5

PORTB = 0x01

#1

PINB = 0x01

#4

PORTB = 0x02

#1

PINB = 0x02

#4

PORTB = 0x03

#1

PINB = 0x03

#4

PORTB = 0x04

#1

PINB = 0x04

#4

PORTB = 0x05

#1

PINB = 0x05

#4

PORTB = 0x06

#1

PINB = 0x06

#4

PORTB = 0x07

#1

PINB = 0x07

#4

PORTB = 0x08

#1

PINB = 0x08

#4

PORTB = 0x09

#1

PINB = 0x09

#4

PORTB = 0x0a

#1

PINB = 0x0a

#4

PORTB = 0x0b

#1

PINB = 0x0b

#4

PORTB = 0x0c

#1

PINB = 0x0c

#4

PORTB = 0x0d

#1

PINB = 0x0d

#4

PORTB = 0x0e

#1

PINB = 0x0e

#4

PORTB = 0x0f

#1

PINB = 0x0f

#4

PORTB = 0x10

#1

PINB = 0x10

#4

PORTB = 0x11

#1

PINB = 0x11

#4

PORTB = 0x12

#1

PINB = 0x12

#4

PORTB = 0x13

#1

PINB = 0x13

#4

PORTB = 0x14

#1

PINB = 0x14

В выходном файле #5 – это количество тактов до изменения отслеживаемого регистра, PORTB = 0x01 – значение регистра после изменения.

Общие требования: задание можно выполнять в режиме пошаговой отладки, но окончательно в виде стимуляции.

**Проверка математических и логических операций, работа с битами**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Мнемоника** | **Операнды** | **Описание** | **Операция** | **Флаги** | **Циклы** |
|  | ADD═ | Rd,Rr═ | Суммирование без переноса | Rd = Rd + Rr═ | Z,C,N,V,H,S═ | 1 |
|  | ADC | Rd,Rr | Суммирование с переносом | Rd = Rd + Rr + C | Z,C,N,V,H,S | 1 |
|  | SUB | Rd,Rr | Вычитание без переноса | Rd = Rd - Rr | Z,C,N,V,H,S | 1 |
|  | SUBI | Rd,K8 | Вычитание константы | Rd = Rd - K8 | Z,C,N,V,H,S | 1 |
|  | SBC | Rd,Rr | Вычитание с переносом | Rd = Rd - Rr - C | Z,C,N,V,H,S | 1 |
|  | SBCI | Rd,K8 | Вычитание константы с переносом | Rd = Rd - K8 - C | Z,C,N,V,H,S | 1 |
|  | AND | Rd,Rr | Логическое И | Rd = Rd ╥ Rr | Z,N,V,S═ | 1 |
|  | ANDI | Rd,K8 | Логическое И с константой | Rd = Rd ╥ K8 | Z,N,V,S | 1 |
|  | OR | Rd,Rr | Логическое ИЛИ | Rd = Rd V Rr | Z,N,V,S | 1 |
|  | ORI | Rd,K8 | Логическое ИЛИ с константой | Rd = Rd V K8 | Z,N,V,S | 1 |
|  | EOR | Rd,Rr | Логическое исключающее ИЛИ | Rd = Rd EOR Rr | Z,N,V,S | 1 |
|  | COM | Rd | Побитная Инверсия | Rd = $FF - Rd | Z,C,N,V,S | 1 |
|  | NEG | Rd | Изменение знака (Доп. код) | Rd = $00 - Rd | Z,C,N,V,H,S | 1 |
|  | SBR | Rd,K8 | Установить бит (биты) в регистре | Rd = Rd V K8 | Z,C,N,V,S | 1 |
|  | CBR | Rd,K8 | Сбросить бит (биты) в регистре | Rd = Rd ╥ ($FF -K8) | Z,C,N,V,S | 1 |
|  | INC | Rd | Инкрементировать значение регистра | Rd = Rd + 1 | Z,N,V,S | 1 |
|  | DEC | Rd | Декрементировать значение регистра | Rd = Rd -1 | Z,N,V,S | 1 |
|  | TST | Rd | Проверка на ноль либо отрицательность | Rd = Rd ╥ Rd | Z,C,N,V,S | 1 |
|  | CLR | Rd | Очистить регистр | Rd = 0 | Z,C,N,V,S | 1 |
|  | SER | Rd | Установить регистр | Rd = $FF | None | 1 |
|  | ADIW | Rdl,K6 | Сложить константу и слово | Rdh:Rdl = Rdh:Rdl +K6═ | Z,C,N,V,S | 2 |
|  | SBIW | Rdl,K6 | Вычесть константу из слова | Rdh:Rdl = Rdh:Rdl -K 6 | Z,C,N,V,S | 2 |
|  | MUL | Rd,Rr | Умножение чисел без знака | R1:R0 = Rd \* Rr | Z,C | 2 |
|  | MULS | Rd,Rr | Умножение чисел со знаком | R1:R0 = Rd \* Rr | Z,C | 2 |
|  | MULSU | Rd,Rr | Умножение числа со знаком с числом беззнака | R1:R0 = Rd \* Rr | Z,C | 2 |
|  | FMUL | Rd,Rr | Умножение дробных чисел без знака | R1:R0 = (Rd \* Rr) <<1 | Z,C | 2 |
|  | FMULS | Rd,Rr | Умножение дробных чисел со знаком | R1:R0 = (Rd \*Rr) <<1 | Z,C | 2 |
|  | FMULSU | Rd,Rr | Умножение дробного числа со знаком счислом без знака | R1:R0 = (Rd \* Rr) <<1 | Z,C | 2 |
|  | LSL | Rd | Логический сдвиг влево | Rd(n+1)=Rd(n), Rd(0)=0,C=Rd(7) | Z,C,N,V,H,S | 1 |
|  | LSR | Rd | Логический сдвиг вправо | Rd(n)=Rd(n+1), Rd(7)=0,C=Rd(0) | Z,C,N,V,S | 1 |
|  | ROL | Rd | Циклический сдвиг влево через C | Rd(0)=C, Rd(n+1)=Rd(n),C=Rd(7) | Z,C,N,V,H,S | 1 |
|  | ROR | Rd | Циклический сдвиг вправо через C | Rd(7)=C, Rd(n)=Rd(n+1),C=Rd(0) | Z,C,N,V,S | 1 |
|  | ASR | Rd | Арифметический сдвиг вправо | Rd(n)=Rd(n+1), n=0,...,6 | Z,C,N,V,S | 1 |
|  | SWAP | Rd | Перестановка тетрад | Rd(3..0) = Rd(7..4), Rd(7..4) =Rd(3..0) | None | 1 |

Задание: Выполнить проверку воздействия различных операций на флаги регистра состояний. Для каждого варианта 5 команд

Варианты

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Вариант | Номера команд | | | | | Вариант | Номера команд | | | | |
| *1* | 3 | 12 | 6 | 9 | 14 | *14* | 33 | 2 | 25 | 28 | 26 |
| *2* | 12 | 29 | 17 | 30 | 10 | *15* | 11 | 29 | 12 | 33 | 31 |
| *3* | 21 | 31 | 29 | 14 | 8 | *16* | 17 | 23 | 21 | 33 | 30 |
| *4* | 26 | 29 | 1 | 9 | 30 | *17* | 27 | 9 | 16 | 18 | 11 |
| *5* | 29 | 9 | 16 | 27 | 10 | *18* | 13 | 31 | 28 | 1 | 23 |
| *6* | 28 | 27 | 18 | 26 | 23 | *19* | 11 | 1 | 12 | 7 | 21 |
| *7* | 6 | 33 | 22 | 3 | 12 | *20* | 13 | 3 | 9 | 26 | 24 |
| *8* | 9 | 7 | 22 | 27 | 2 | *21* | 12 | 34 | 32 | 21 | 5 |
| *9* | 4 | 24 | 30 | 14 | 27 | *22* | 2 | 25 | 6 | 23 | 3 |
| *10* | 13 | 26 | 24 | 11 | 9 | *23* | 23 | 25 | 3 | 31 | 29 |
| *11* | 22 | 27 | 25 | 4 | 7 | *24* | 12 | 28 | 27 | 32 | 31 |
| *12* | 3 | 14 | 6 | 18 | 31 | *25* | 26 | 18 | 1 | 6 | 21 |
| *13* | 9 | 4 | 29 | 24 | 17 | *26* | 6 | 27 | 23 | 4 | 5 |

**Пример выполнения для команды Add**

Мнемоника ADD

Операнды Rd,Rr

Описание Суммирование без переноса

Операция Rd = Rd + Rr

Флаги Z,C,N,V,H,S

Циклы 1

Задача: проверить изменение флагов при различных значениях операндов.

**Текст программы**

reset:

rjmp main

main:

; загрузка значений в регистры

ldi r18, 0xFF

ldi r19, 0xFF

; вывод данных из РОН в IO регистр для отображения

out OCR0A, r18

out OCR0B, r19

nop

loop:

; ввод данных из IO регистров в РОН для обработки

; так как арифм и логические инструкции работают с РОН

in r18, OCR0A

in r19, OCR0B

; выполнение операции

add r18, r19

; вывод данных из РОН в IO регистр для отображения

out OCR0A, r18

out OCR0B, r19

rjmp loop

Для создание теста необходимо знать количество тактов до loop, значения при которых будут изменятся флаги. Например, флаг H изменится при суммировании 15 и числом от 1 до 240, так как произойдёт переход в старший полубайт.

**Тестовый файл будет выглядеть следующим образом:**

$log OCR0A

$log OCR0B

$log SREG

$startlog Asm\_Lab\_Math\_log\_output.stim

#6

OCR0A = 0

OCR0B = 1

#7

OCR0A = 1

OCR0B = 15

#7

OCR0A = 110

OCR0B = 16

#7

OCR0A = 1

OCR0B = 126

#7

OCR0A = 1

OCR0B = 127

#7

OCR0A = 126

OCR0B = 128

#7

OCR0A = 1

OCR0B = 254

#7

OCR0A = 1

OCR0B = 255

#7

$stoplog

$break

**Здесь:**

$log OCR0A

$log OCR0B

$log SREG

$startlog mega328p\_log\_output.stim

Включение логирования регистров IO и запись в файл

#6

OCR0A = 0

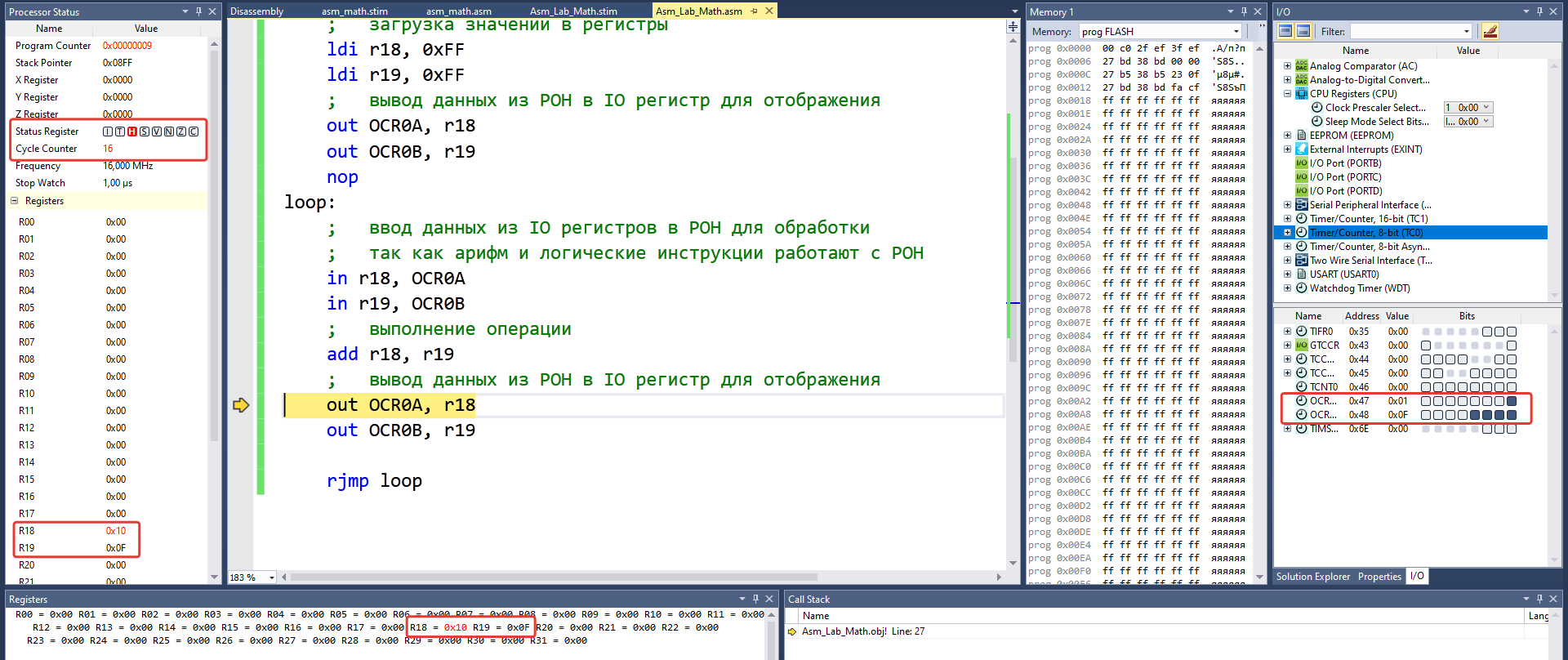
OCR0B = 1

Далее задержка на 6 тактов до метки loop и установка значений в регистры OCR0A и OCR0B

#7

Задержка на 7 тактов до следующей итерации loop.

Внимание! В выходном файле регистры будут отображаться только при изменении значения в них, поэтому тестовые значения подобраны таким образом, чтобы на следующей итерации loop значения были изменены.



Результат в пошаговом режиме

**Выходной файл:**

#3

OCR0A = 0xff

#1

OCR0B = 0xff

#2

OCR0B = 0x01

OCR0A = 0x00

#3

OCR0A = 0x01

#4

OCR0B = 0x0f

#2

SREG = 0x20

#1

OCR0A = 0x10

#4

OCR0B = 0x10

OCR0A = 0x6e

#2

SREG = 0x00

#1

OCR0A = 0x7e

#4

OCR0B = 0x7e

OCR0A = 0x01

#3

OCR0A = 0x7f

#4

OCR0B = 0x7f

OCR0A = 0x01

#2

SREG = 0x2c

#1

OCR0A = 0x80

#4

OCR0B = 0x80

OCR0A = 0x7e

#2

SREG = 0x14

#1

OCR0A = 0xfe

#4

OCR0B = 0xfe

OCR0A = 0x01

#3

OCR0A = 0xff

#4

OCR0B = 0xff

OCR0A = 0x01

#2

SREG = 0x23

#1

OCR0A = 0x00

**Разбор файла:**

Через три такта после запуска программы OCR0A = 0xff, затем через 1 такт OCR0B = 0xff.

Еще через 2 такта значения регистров изменены из тестового файла, далее 3такта на запись значений в r18 и r19 и один такт на выполнение сложения с последующим переносом значения в OCR0A. 4 Такта на запись в IO регистры и переход на новую итерацию.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | OCR0A | OCR0B | SREG | OCR0A’ | Флаги |
|  | 0x00 | 0x01 | 0x00 | 0x01 | - |
|  | 0x01 | 0x0f | 0x20 | 0x10 | H |
|  | 0x6e | 0x10 | 0x00 | 0x7e | - |
|  | 0x01 | 0x7e | 0x00 | 0x7f | - |
|  | 0x01 | 0x7f | 0x2c | 0x80 | H,V,N |
|  | 0x7e | 0x80 | 0x14 | 0xfe | S,N |
|  | 0x01 | 0xfe | 0x14 | 0xff | S,N |
|  | 0x01 | 0xff | 0x23 | 0x00 | H,Z,C |

В выводах по разделу отразить особенности изменения флагов в зависимости от значений.

**Условные переходы**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Мнемоника** | **Операнды** | **Описание** | **Операция** | **Флаги** | **Циклы** |
|  | RJMP | k | Относительный переход | PC = PC + k +1 | None | 2 |
|  | IJMP | Нет | Косвенный переход на (Z) | PC = Z | None | 2 |
|  | EIJMP | Нет | Расширенный косвенный переходна (Z) | STACK = PC+1, PC(15:0) = Z,PC(21:16) = EIND | None | 2 |
|  | JMP | k | Переход | PC = k | None | 3 |
|  | RCALL | k | Относительный вызовподпрограммы | STACK = PC+1, PC = PC + k +1 | None | 3/4\* |
|  | ICALL | Нет | Косвенный вызов (Z) | STACK = PC+1, PC = Z═ | None | 3/4\* |
|  | EICALL | Нет | Расширенный косвенный вызов (Z) | STACK = PC+1, PC(15:0) = Z,PC(21:16) =EIND | None | 4\* |
|  | CALL | k | Вызов подпрограммы | STACK = PC+2, PC = k | None | 4/5\* |
|  | RET | Нет | Возврат из подпрограммы | PC = STACK | None | 4/5\* |
|  | RETI | Нет | Возврат из прерывания | PC = STACK | I | 4/5\* |
|  | CPSE | Rd,Rr | Сравнить, пропустить если равны═ | if (Rd ==Rr) PC = PC 2 or 3 | None | 1/2/3 |
|  | CP | Rd,Rr | Сравнить | Rd -Rr | Z,C,N,V,H,S | 1 |
|  | CPC | Rd,Rr | Сравнить с переносом | Rd - Rr - C | Z,C,N,V,H,S | 1 |
|  | CPI | Rd,K8 | Сравнить с константой | Rd - K | Z,C,N,V,H,S | 1 |
|  | SBRC | Rr,b | Пропустить если бит в регистреочищен | if(Rr(b)==0) PC = PC + 2 or 3 | None | 1/2/3 |
|  | SBRS | Rr,b | Пропустить если бит в регистреустановлен | if(Rr(b)==1) PC = PC + 2 or 3 | None | 1/2/3 |
|  | SBIC | P,b | Пропустить если бит в портуочищен | if(I/O(P,b)==0) PC = PC + 2 or 3 | None | 1/2/3 |
|  | SBIS | P,b | Пропустить если бит в портуустановлен | if(I/O(P,b)==1) PC = PC + 2 or 3 | None | 1/2/3 |
|  | BRBC | s,k | Перейти если флаг в SREGочищен | if(SREG(s)==0) PC = PC + k + 1 | None | 1/2 |
|  | BRBS | s,k | Перейти если флаг в SREGустановлен | if(SREG(s)==1) PC = PC + k + 1 | None | 1/2 |
|  | BREQ | k | Перейти если равно | if(Z==1) PC = PC + k + 1 | None | 1/2 |
|  | BRNE | k | Перейти если не равно | if(Z==0) PC = PC + k + 1 | None | 1/2 |
|  | BRCS | k | Перейти если перенос установлен | if(C==1) PC = PC + k + 1 | None | 1/2 |
|  | BRCC | k | Перейти если перенос очищен | if(C==0) PC = PC + k + 1 | None | 1/2 |
|  | BRSH | k | Перейти если равно или больше | if(C==0) PC = PC + k + 1 | None | 1/2 |
|  | BRLO | k | Перейти если меньше | if(C==1) PC = PC + k + 1 | None | 1/2 |
|  | BRMI | k | Перейти если минус | if(N==1) PC = PC + k + 1 | None | 1/2 |
|  | BRPL | k | Перейти если плюс | if(N==0) PC = PC + k + 1 | None | 1/2 |
|  | BRGE | k | Перейти если больше или равно(со знаком) | if(S==0) PC = PC + k + 1 | None | 1/2 |
|  | BRLT | k | Перейти если меньше (со знаком) | if(S==1) PC = PC + k + 1 | None | 1/2 |
|  | BRHS | k | Перейти если флаг внутреннегопереноса установлен | if(H==1) PC = PC + k + 1 | None | 1/2 |
|  | BRHC | k | Перейти если флаг внутреннегопереноса очищен | if(H==0) PC = PC + k + 1 | None | 1/2 |
|  | BRTS | k | Перейти если флаг T установлен | if(T==1) PC = PC + k + 1 | None | 1/2 |
|  | BRTC | k | Перейти если флаг T очищен | if(T==0) PC = PC + k + 1 | None | 1/2 |
|  | BRVS | k | Перейти если флаг переполненияустановлен | if(V==1) PC = PC + k + 1 | None | 1/2 |
|  | BRVC | k | Перейти если флаг переполненияочищен | if(V==0) PC = PC + k + 1 | None | 1/2 |
|  | BRIE | k | Перейти если прерыванияразрешены | if(I==1) PC = PC + k + 1 | None | 1/2 |
|  | BRID | k | Перейти если прерываниязапрещены | if(I==0) PC = PC + k + 1 | None | 1/2 |

Задание: Написать программу с использованием условного перехода. Для каждого варианта 5 команд

Варианты

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Вариант | Номера команд | | | | | Вариант | Номера команд | | | | |
| *1* | 23 | 18 | 29 | 32 | 28 | *14* | 34 | 31 | 14 | 36 | 21 |
| *2* | 30 | 20 | 25 | 23 | 16 | *15* | 31 | 11 | 33 | 36 | 20 |
| *3* | 22 | 13 | 11 | 30 | 28 | *16* | 14 | 32 | 26 | 24 | 16 |
| *4* | 13 | 16 | 33 | 36 | 15 | *17* | 27 | 19 | 15 | 29 | 16 |
| *5* | 32 | 24 | 19 | 29 | 34 | *18* | 16 | 12 | 19 | 23 | 15 |
| *6* | 28 | 16 | 31 | 12 | 33 | *19* | 22 | 24 | 11 | 20 | 19 |
| *7* | 21 | 22 | 14 | 36 | 26 | *20* | 33 | 23 | 29 | 16 | 30 |
| *8* | 26 | 33 | 19 | 35 | 12 | *21* | 30 | 11 | 36 | 33 | 13 |
| *9* | 19 | 36 | 26 | 28 | 16 | *22* | 35 | 24 | 28 | 20 | 34 |
| *10* | 20 | 14 | 21 | 16 | 29 | *23* | 22 | 15 | 25 | 12 | 35 |
| *11* | 11 | 33 | 28 | 29 | 14 | *24* | 20 | 36 | 35 | 28 | 23 |
| *12* | 34 | 28 | 13 | 35 | 32 | *25* | 13 | 15 | 28 | 12 | 33 |
| *13* | 21 | 34 | 31 | 13 | 33 | *26* | 34 | 31 | 14 | 36 | 21 |

Пример для оператора SBRC

Мнемоника SBRC

Операнды Rr,b

Описание Команда проверяет состояние бита в регистре и, если этот бит очищен, пропускает следующую команду.

Операция if(Rr(b)==0) PC = PC + 2 or 3

Флаги None

Циклы 1/2/3

Счетчик программ: PC <-- PC + 1, если условия не соблюдены, нет пропуска; PC <-- PC + 2, если следующая команда длиной в 1слово; PC <-- PC + 3, если следующие команды JMP или CALL

Код программы:

reset:

rjmp main

main:

nop

loop:

; ввод данных из IO регистров в РОН для обработки

in r18, OCR0A

; выполнение операции

ldi r19, (1 << 4)

sbrc r18, 4

ldi r19, 1

; вывод данных из РОН в IO регистр для отображения

out OCR0B, r19

rjmp loop

Код теста:

$log OCR0A

$log OCR0B

$startlog Asm\_Lab\_Branch\_log\_output.stim

#2

OCR0A = 0x10

#7

OCR0A = 0x41

#7

$stoplog

$break

Выходной файл:

#2

OCR0A = 0x10

#4

OCR0B = 0x01

#3

OCR0A = 0x41

#4

OCR0B = 0x10

|  |  |  |
| --- | --- | --- |
|  | OCR0A | OCR0B |
| 1. | 0x10 | 0x01 |
| 2. | 0x41 | 0x10 |

Вывод: при установленном 4м бите следующая команда выполняется, при сброшенном – пропуск и счетчик циклов увеличен на 2. Если после команды SBRC идет команда в один цикл, то общее количество циклов сохраняется.

**Работа с массивами**

Задачи:

1. Пузырьковая сортировка
2. Сортировка выбором
3. Сортировка вставкой
4. Ряд Фиббоначи до 35 элементов
5. Контрольная сумма через XOR
6. Преобразование массива (инверсия порядка бит)
7. Формирование массива путем выбора ячеек из другого dest[i] = src[tmp[i]]

Задание: выбрать задачу в соответствии с вариантом. Написать программу и тест.

Критерий оценки: 3 баллов для типа байт, 5 баллов – знаковый байт, 6 баллов – слово или знаковое слово

Для примера программа копирования массива с инверсией байт

.set ARR\_SIZE=10

; В сегменте данных выделим массив типа BYTE

.dseg

arr: .BYTE ARR\_SIZE

.cseg

reset:

rjmp main

main:

ldi ZH,High(src\*2) ;загрузка адреса 0-го

ldi ZL,Low(src\*2) ;элемента в рег. пару Z

ldi YH,High(arr) ;загрузка адреса 0-го

ldi YL,Low(arr) ;элемента в рег. пару Y

ldi R18, ARR\_SIZE ; загрузка размера массива в счётчик элементов массива

arr\_copy:

lpm R0, Z+ ; Загрузка байта по адресу Z в регистр R0 с увеличением адреса

out OCR0A, R0 ; вывод значения в регистр IO

com R0 ; побитная инверсия значения в регистре R0

out OCR0B, R0 ; вывод значения в регистр IO

st Y+, R0 ; Сохранение значения регистра R0 байта в ОЗУ по адресу Y с увеличением адреса

subi R18, 1 ; Уменьшение значения счётчика итераций

brne arr\_copy ; Условный переход пока не прошли все элементы массива

loop:

nop

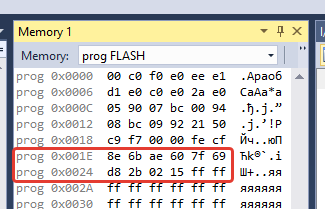
rjmp loop

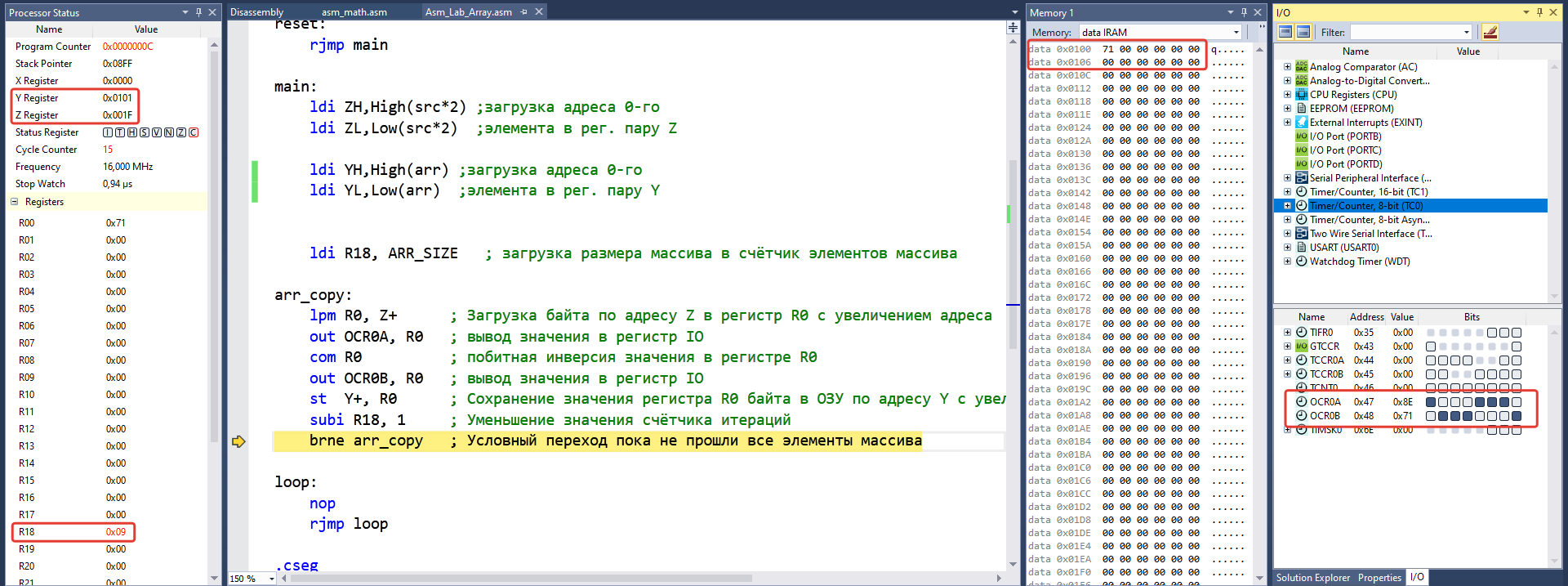
.cseg

; В сегменте кода выделим массив типа BYTE

src: .db 0x8E, 0x6B, 0xAE, 0x60, 0x7F, 0x69, 0xD8, 0x2B, 0x02, 0x15

ldi ZH,High(src\*2) Этой командой мы загружаем в старшую часть пары Z (ZH), старшую часть адреса по метке src. Что значит "\*2"? Дело в том, что каждая команда содержит два байта информации и занимает, таким образом, две ячейки ПЗУ. Поэтому, счетчик команд считает 2 адреса как один. Метка содержит именно данные для счетчика команд. Чтобы получить реальный адрес ПЗУ, необходимо увеличить адрес метки в 2 раза.





Файл стимуляции:

$log OCR0A

$log OCR0B

$log SREG

$startlog Asm\_Lab\_Array\_log\_output.stim

#115

$stoplog

$memdump Asm\_Lab\_Array\_mem\_S\_log\_output.stim 0x0100 10 s

$memdump Asm\_Lab\_Array\_mem\_F\_log\_output.stim 0x001E 10 f

$break

Обратить внимание на состояние памяти данных и ОЗУ. На копирование массива с инверсией потребовалось 115 циклов. Разобрать и добавить дамп памяти в отчёт в hex и в dec.

reset:

rjmp main

main:

ldi ZH,High(arr) ;загрузка адреса 0-го

ldi ZL,Low(arr) ;элемента в рег. пару Z

ldi r16, 0

ldi r17, 1

st Z+, r16

st Z, r17

ldi R18, ARR\_SIZE ; загрузка размера массива в счётчик элементов массива

nop

arr\_copy:

ld r17, Z+ ; ld - берёт данные из памяти

add r16, r17 ; сложение рег.17 и рег.16(r16 + r17)

st Z, r16 ; Запись в память по адресу Z значения в рег.17

mov r16, r17

subi R18, 1 ; Уменьшение значения счётчика итераций

brne arr\_copy ; Условный переход пока не прошли все элементы массива