AVR არქიტექტურის მიკროკონტროლერებისთვის პროგრამული უზრუნველყოფის შემუშავება გრაფიკულ გარემოში Algorithm Builder

ieeetsu.ge
2012

სარჩევი

AVR არქიტექტურის მიკროკონტროლერებისთვის პროგრამული უზრუნველყოფის შემუშავება გრაფიკულ გარემოში Algorithm Builder.....1
სახელმძღვანელოს შესახებ.....2
გარემოს შესახებ.....3
აღგორითმის აგებულება.....3
აღგორითმის ელემენტები.....3
ელემენტი FIELD – ველი.....3
ელემენტი LABEL – ჭდე.....4
ელემენტი VERTEX – ბღოკის წვერო.....5
ელემენტი CONDITION – პირობითი გადასვლა.....5
ელემენტი JMP Vector – უთუო გადასვლა.....6
ელემენტი SETTER – მიმნიჭებელი.....6
ელემენტი TEXT - ადგილობრივი ტექსტური რედაქტორის სტრიქონი.....8
პროექტის ფაილები.....8
გარემოს აღწერა.....9
მარტივი აღგორითმის მაგალითი.....9
მიკროკონტროლერის ოპერატორები.....13
კოპირების და არითმეტიკა-ლოგიკური გარდაქმნების ქმედებები.....13
პირობითი გადასვლის ოპერატორები.....17
შემდეგი ოპერატორის პირობითი გამოტოვების ოპერატორები.....17
კომენტარები ოპერატორებთან დაკავშირებით.....18
მუდმივების წარდგენა უშუალი სახით.....19
რესურსების განაწილება და სახელების გამოცხადება.....19
ცვლადთა ფორმატები.....20
მუდმივების გამოცხადების განყოფილება.....20
სამუშაო რეგისტრთა სახელების გამოცხადების განყოფილება.....21
I\O რეგისტრთა სახელების გამოცხადების განყოფილება.....21
ბიტების სახელების გამოცადება.....22
SRAM ცვლადების გამოცხადების განყოფილება.....22
EEPROM ცვლადების გამოცხადების განყოფილება.....23
მუდმივების წარდგენა აღგებრული ჩანაწერების სახით.....25
მაკრო-ოპერატორები.....26
პირობითი ოპერატორები.....31
წყვეტის მომსახურე ბღოკები.....31
მონაცემთა პირდაპირი განთავსება პროგრამის მეხსიერებაში.....32
მონაცემთა ფაილის პროგრამის მეხსიერებაში ჩართვა.....33
პროექტში აღგორითმის ჩართვა გარე ფაილებიდან.....33
მაგალითი - ვოლტმეტრი.....34
აღგორითმის დამუშავება.....36
რესურსთა განაწილების ცხრილის დამუშავება.....39
მიკროკონტროლერის დაპროგრამება.....40
აღგორითმის მუშაობის გამართვა სიმულატორში.....41
აღგორითმის მუშაობის გამართვა კრისტალზე (მონიტორული გამარათვა).....43
აქტიური გადამყვანი დაპროგრამებისთვისა და მონიტორული გამართვისთვის...46
პირობითი დაკომპილება.....47
მომხმარებლის მაკროსები.....48
პარამეტრებიანი ქვეპროგრამები.....49
მონაცემების ფორმატი მოცურავე წერტილით (ათწილადები).....50
ჩამტვირთავის დაპროგრამება.....51
ინფორმაცია შეცვლილი სამუშაო რეგისტრების შესახებ.....53
გამოყენებული ტერმინოლოგიის მოკლე სიტყვარი.....54

სახელმძღვანელოს შესახებ

თქვენს წინაშე მყოფი სახელმძღვანელო წარმოადგენს Algorithm Builder-ის ე.წ. 'manual'-ის სამოყვარულო თარგმანს. ნათარგმნია როგორც ტექსტი, ისე ილუსტრაციები, რაც შეიძლება გამართული ქართული ენით.

თარგმანის ავტორმა შეეცადა რაც შეიძლება ნაკლები უცხო სიტყვა გამოეყენა, რის გამოც შეიძლება ზოგიერთი ტერმინი უცხოდ მოეჩვენოს მკითხველს. ამისთვის, სახელმძღვანელოს ბოლოს განთავსებულია მოკლე სიტყვარი-განმარტებითი ლექსიკონი, სადაც მოყვანილია ზოგიერთი ქართული ტერმინი, მათი ინგლისური და რუსული შესაბამისი სიტყვები და მიახლოებითი განმარტება.

წარმატებას გისურვებთ!

- გ.

* * *

სახელმძღვანელოს შექმნისთვის გამოყენებული იყო მხოლოდ თავისუფალი და ღია-წყაროს მქონე პროგრამული უზრუნველყოფა (free and open-source software):

საოპერაციო სისტემა: **Ubuntu GNU \Linux 11.10**

საოფისე კრებული: **Libre Office 3.5.0**

გამოსახულებათა დამუშავებისთვის: **GIMP 2.7**

დოკუმენტში გამოყენებულია შემდეგი შრიფტები:

BPG SuperSquare, DejaVu Sans, DejaVu Sans Mono

უღრმესი მადლობა ყველა იმ შესანიშნავ ადამიანს,
ვისაც წვლილი შეაქვს ამ პროდუქტების შემუშავებაში!

გარემოს შესახებ

მოცემული გარემო უზრუნველყოფს შემუშავების სრულ ციკლს, დაწყებული ალგორთმის შეყვანითა და გამართვით, დამთავრებული კრისტალის შიგასქემატური დაპროგრამებით. გექნებათ საშუალება შეიმუშავოთ პროგრამები როგორც ასემბლერის(assembly) დონეზე, ასევე მაკრო-დონეზე, რომელშიც შესაძლებელია ნებისმიერი სიგრძის მქონე ნამდვილ რიცხვებთან *(ათწილადებთან)* მუშაობა. ეს მაქსიმალურად აახლოვებს დაპროგრამების შესაძლებლობებს მაღალი დონის ენებთან.

პროგრამების შემუშავების გრაფიკული ტექნოლოგიები პროგრამისტებისტვის ახალ შესაძლებლობებს აჩენენ. მათ ეძლევათ საშუალება შეიყვანონ პროგრამა სიბრტყეზე – *ხისებრი* სტრუქტურის მქონე ალგორითმის სახით. ამის შედეგად პროგრამის ლოგიკური სტრუქტურა თვალსაჩინო ხდება. ამ ტექნოლოგიების ძირითადი მიზანია შემუშავების იერსახის ადამიანის მიერ ბუნებრივად აღქმა. ასეთი გარემოს შესწავლა გაცილებით მარტივია ვიდრე კლასიკური *ასემბლერის* შესწავლა. უფრო მოსახერხებელი იერსახე შემუშავების ახალ ჰორიზონტებს ხსნის. მომხმარებელთა შეფასებით, პროგრამული უზრუნველყოფის შემუშავების დრო მცირდება 3 - 5 ჯერ კლასიკურ *ასემბლერთან* შედარებით.

ეს გარემო განკუთვნილია საოპრაციო სისტემისთვის Windows 95/98/2000/NT/ME/XP რედაქტორის გამართულად მუშაობისათვის საჭიროა სისტემაში იყოს ჩაყენებული შრიფტი «Courier»

ალგორითმის აგებულება

ნებისმიერი პროგრამა შესაძლოა დაიყოს ცალკე მდგომ ლოგიკურად დასრულებულ ნაწილებად. როგორც წესი, ამ ნაწილების დამასრულებელი ოპერატორად გამოიყენება *უთუო გადასვლა* ან ქვეპროგრამიდან დაბრუნება, ანუ ოპერატორები, რომელთ შემდეგ პროგრამის წრფივი შესრულება ცალსახად წყდება.

Algorithm Builder გარემოში პროგრამის შემუშავების პროცესი დაიყვანება ასეთი ბლოკების შედგენაზე, სიბრტყეზე განლაგება და მათ შორის ვექტორული კავშირის დამყარება პირობითი და უთუო გადასვლებით.

ალგორითმის ელემენტები

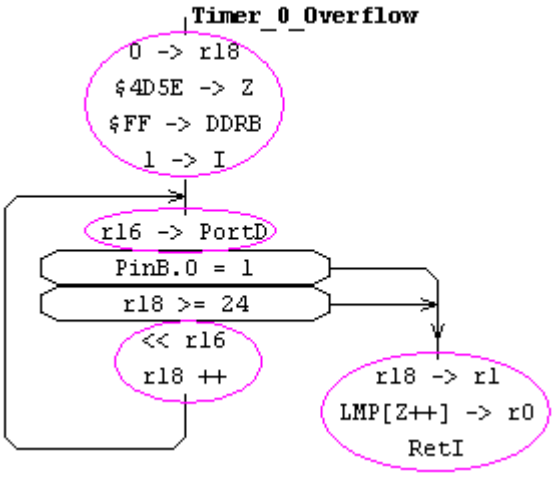
ალგორითმების შედგენისთვის გათვალისწინებულია შვიდი ელემენტი:

FIELD	-	ველი;
LABEL	-	ჭდე;
VERTEX	-	ბლოკის წვერო;
CONDITION	-	პირობითი გადასვლა;
JMP Vector	-	ფარდობითი უთუო გადასვლა;
SETTER	-	პერიფერიული მოწყობილობების მიმნიჭებელი;
TEXT	-	ადგილობრივი ტექსტური რედაქტორის სტრიქონი;

ელემენტი FIELD – ველი

წარმოადგენს ბლოკის ცენტრში მყოფ სტრიქონს. ელემენტი გამოიყენება მიკროკონტროლერის ოპერატორების უმეტესობის ჩაწერისთვის. ველის დასამატებლად გამოიყენეთ მენიუს ელემენტი “Elements\Field”, ან დააჭირეთ ხელსაწყოთა პანელზე მყოფ ღილაკს F, ან კლავიატურის ღილაკთა კომბინაციას “Alt+F”, ან კლავიატურის ღილაკს “Enter” (თუ კუსრსორი მდებარეობს ლოკალური ტექსტური რედაქტორის გარეთ).

ქვემოთმოცემულ მაგალითში FIELD ტიპის ელემენტები ოვალებშია მოთავსებული:

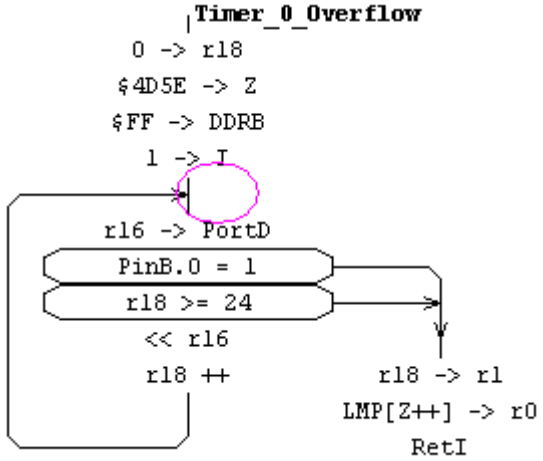


ელემენტი LABEL – ჭდე

წარმოადგენს ვერტიკალურ ოპერატორთა ბლოკში მოთავსებულ *შტრიხს* და არააუცილებელ დასახელებას, რომელიც მოთავსებულია შტრიხის მარცხენა ან მარჯვენა მხარეს. გამოიყენება ალგორითმში ისეთი ადგილების მოსანიშნად, სადაც შესაძლებელი იქნება პირობითი ან უთუო გადასვლების განხორციელება. ჭდის ბლოკში ჩასამატებლად გამოიყენეთ მენიუს ელემენტი “Elements\Label”, ან დააჭირეთ ხელსაწყოთა პანელზე მყოფ ღილაკს  , ან კლავიატურის ღილაკთა კომბინაციას “Alt+L”. საჭიროების შემთხვევაში შესაძლებელია მიუთითოთ პროგრამის კონკრეტული მისამართი. ამისთვის, დასახელების წინ (თუ ის არსებობს) უნდა შეიყვანოთ მუდმივა ან ალგებრული ჩანაწერი, რომელიც განსაზღვრავს სასურველ მისამართს.

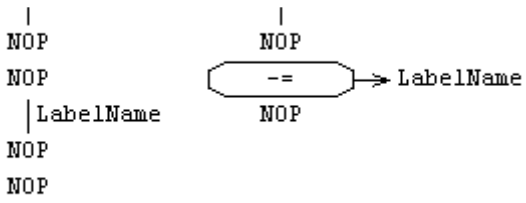
რათა შეცვალოთ ჭდის წარწერის მდებარეობა საწინააღმდეგოთი, დააჭირეთ ღილაკს “Tab”.

ქვემოთმოცემულ მაგალითში LABEL ტიპის ელემენტები ოვალებშია მოთავსებული:



როგორც წესი, ჭდეზე უნდა იყოს მიმაგრებული რომელიმე გადასვლის ვექტორი (ისრით ბოლოში). ამ შემთხვევაში ჭდისთვის სახელის მიცემა არ არის აუცილებელი. Algorithm Builder იძლევა საშუალებას გამოიყენოთ გადასვლების კლასიკური გადამისამართება ჭდეების დასახელების გამოყენებით. ამ შემთხვევაში სახელის დარქმევა *აუცილებელია*.

მაგალითისთვის:

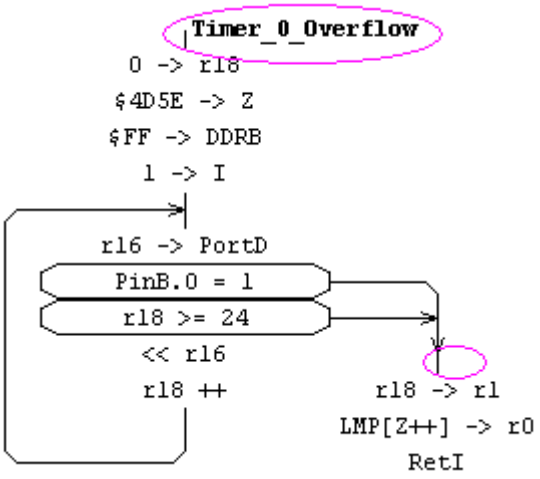


გარდა ამისა, ალგორითმების ბლოკებში ჭდის დასახელება შეიძლება იყოს გამოყენებული მუდმივის სახით, რომელიც შეიცავს სათანადო მისამართს პროგრამის მეხსიერებიდან.

ელემენტი VERTEX – ბლოკის წვერო

მისი გამოსახულებითა და დანიშნულებით ჭდის აბსოლუტური იდენტურია, მაგრამ ერთი განსხვავებით – ის განსაზღვრავს ბლოკის მდებარეობას სიბრტყეზე და ყოველთვის ძევს მის დასაწყისში. რათა დაამატოთ ახალი წვერო გამოიყენთ მენიუს ელემენტი “Elements\Vertex” ან დააჭირეთ ხელსაწყოთა პანელზე მყოფ ღილაკს V, ან კლავიატურის ღილაკთა კომბინაციას “Alt+V”, ან თაგვის მარცხენა ღილაკის წკაპვით სამუშაო არის სასურველ ადგილას კლავიატურის ღილაკთა კომბინაციასთან ერთად “Ctrl + Alt +Shift”.

ქვემოთმოცემულ მაგალითში VERTEX ტიპის ელემენტები ოვალებშია მოთავსებული:



როგორც წესი, წვეროს ენიჭება დასახელება მხოლოდ როცა ის წარმოადგენს ქვეპროგრამის ან მაკროსის დასაწყისს.

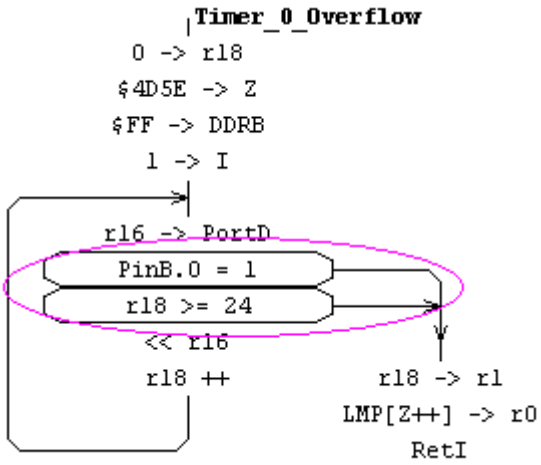
ელემენტი CONDITION – პირობითი გადასვლა

კონსტრუქციის მიხედვით ეს ელემენტი გაცილებით რთულია – ის განკუთვნილია პირობითი გადასვლისთვის. წარმოადგენს ოვალურ კონტურს, რომელშიც ჩაწერილია გადასვლის პირობა და ტეხილი ხაზის სახით გადასვლის სავარაუდო ვექტორი. ხაზის ბოლოში მოთავსებულია ისარი, სადაც მითითებულია ვექტორის არააუცილებელი დასახელება.

ვექტორის ბოლოში მყოფი ისარი უნდა მიუთითებდეს ან რომელიმე ჭდეს, ან წვეროს, ან სხვა ვექტორს, ან ჰქონდეს სამიზნე ჭდის დასახელება (რომელზეც ხდება გადასვლა).

რათა მიმართოთ ვექტორი სასურველ ადგილას, დააჭირეთ და გეჭიროთ ღილაკი “Alt” და დაწკაპეთ თაგვის მარცხენა ღილაკით. ვექტორის ჩასწორებისთვის “Alt” ღილაკთან ერთად გამოიყენეთ ისრიანი ღილაკები. რათა გადახვიდეთ ვექტორის პირობის ჩასწორების რეჟიმიდან ვექტორის დასახელების ჩასწორების რეჟიმზე, დააჭრეთ ღილაკს “Tab”. იგივე ღილაკის მომდევნო დაჭერა შეცვლის ვექტორის მიმართულებას საწინააღმდეგოთი. რათა დაამატოთ ახალი პირობითი გადასვლა გამოიყენეთ მენიუს ელემენტი “Elements\Condition”, ან დააჭირეთ ხელსაწყოთა პანელზე მყოფ ღილაკს C, ან კლავიატურის ღილაკთა კომბინაციას “Alt+C”.

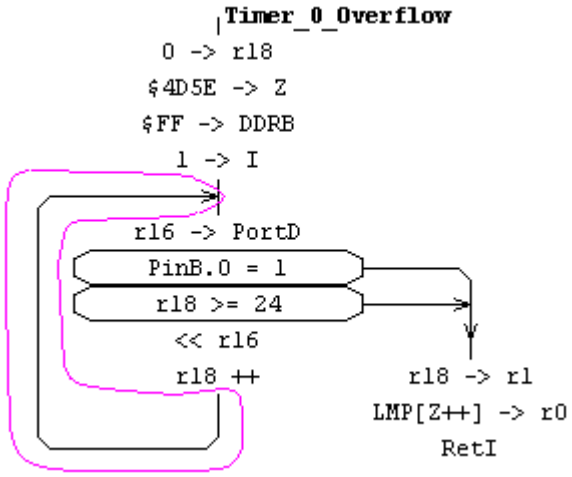
ქვემოთ მოცემულ მაგალითში CONDITION ტიპის ელემენტები ოვალებშია მოთავსებული:



ელემენტი JMP Vector – უთუო გადასვლა

ელემენტი გამოიყენება *პირდაპირი გადასვლებისთვის* (კლასიკურ ასემბლერში ეს ოპერატორი არის "RJMP"). ის წარმოადგენს ტეხილ ხაზს, რომელიც გამოდის ბლოკის ტანიდან და მთავრდება ისეთივე ისრით, როგორც ელემენტი CONDITION. რათა დაამატოთ უთუო გადასვლა, გამოიყენეთ მენიუს ელემენტი "Elements\JPM Vector", ან დააჭირეთ ხელსაწყოთა პანელზე მყოფ ღილაკს J, ან კლავიატურის ღილაკთა კომბინაციას "Alt+J".

ქვემოთ მოცემულ მაგალითში JMP Vector ტიპის ელემენტები ოვალებშია მოთავსებული:



ელემენტი SETTER – მიმნიჭებელი

ეს ელემენტი წარმოადგენს რუხ მართკუთხედს, რომელშიც ჩაწერილია მის მიერ გასამართი პერიფერიული კომპონენტი, როგორიცაა მთვლელი ანალოგურ-ციფრული გარდამქმნელი (**აცგ**), შეწყვეტის ნიღბის რეგისტრი და სხვა. მიმნიჭებელი გამოიყენება მიკროკონტროლერის ისეთი ქმედებათა მიმდევრობის შესაქმნელად, რომლებიც უზრუნველყოფენ საჭირო მუდმივების ჩატვირთვას შეყვანა-გამოტანის სათანადო რეგისტრებში, მითითებული თვისებების გათვალისწინებით.

ამ ელემენტის გამოყენების წინ აუცილებლად უნდა იყოს განსაზღვრული მიკროკონტროლერის ტიპი (მენიუს ელემენტი "Options\Project Options", ჩანართი "Chip")

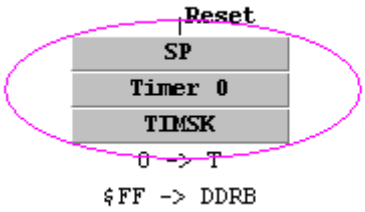
მიმნიჭებლის დამატებისთვის გამოიყენეთ მენიუს ელემენტი "Elements\Setter", ან დააჭირეთ ხელსაწყოთა პანელზე მყოფ ღილაკს S, ან კლავიატურის ღილაკთა კომბინაციას "Alt+S". უკვე არსებული მიმნიჭებლის ჩასასწორებლად, გააქტიურეთ რედაქტორი ორმაგი წკაპით ან კლავიატურის ღილაკთა კომბინაციით "Shift+Enter".

ელემენტი არის მაკრო-ოპერატორი. დაკომპილების შემდეგ ის გარდაიქმნება მიკროკონტროლერის ისეთ ბრძანებათა მიმდევრობად, რომელიც უზრუნველყოფს საჭირო მუდმივების ჩატვირთვას შეყვანა\გამოტანის სათანადო მმართველ რეგისტრებში. ამასთან,

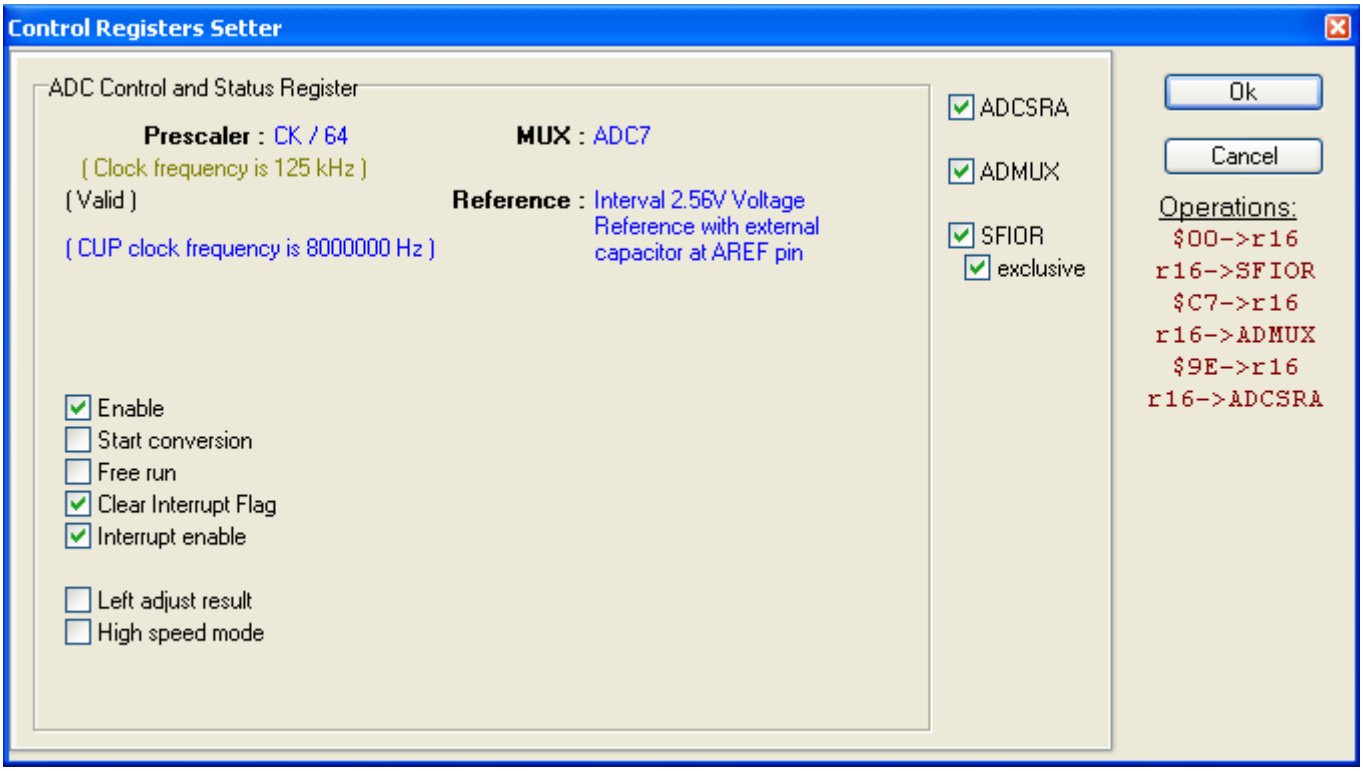
უნდა გაითვალისწინოთ, რომ ამ ოპერაციებში გამოყენებული იქნება რეგისტრი-შუამავალი r16.

კომპონენტთა მთელი რიგისთვის, მაგალითად **აცგ**, მიმნიჭებელს შეუძლია იქონიოს ზეგავლენა რამდენიმე მმართველ რეგისტრზე. ამ შემთხვევაში, საჭიროების მიხედვით შესაძლოა დაიბლოკოს ზემოქმედება კონკრეტულ რეგისტრებზე.

ქვემოთმოცემულ მაგალითში SETTER ტიპის ელემენტები ოვალებშია მოთავსებული:



აცგ მიმნიჭებლის გახსნილი ფანჯარის მაგალითი Atmega128 მიკრ-თვის:



მარჯვენა მხარეს, ღილაკების ქვეშ, ნაჩვენებია მიმნიჭებლის მიერ წარმოქმნილი ოპრაციები. სამუშაო გვერდის მარჯვენა მხარეს ვერტიკალურ სვეტში მოთავსებულია მმართველი რეგისტრების საქტივაციო გასაღებები, რომლებიც საშუალებას იძლევიან ზეგავლენა მოახდინოთ კონკრეტულ რეგისტრებზე.

ზოგიერთ გასაღებს გააჩნია პარამეტრი “exclusive”. მისი არსებობა ნიშნავს, რომ ეს რეგისტრი გამოიყენება არა მხოლოდ მიმდინარე კომპონენტის მიერ (მოყვანილ მაგალითში რეგისტრი “SFIOR” გამოიყენება არა მხოლოდ **აცგ**-ს მიერ). ამასთან, თუ ეს პარამეტრი ჩართულია, მაშინ უქმ ბიტებში კომპონენტი(ამ მაგალითში – **აცგ**) ჩაწერს ნულებს. პარამეტრის გათიშვის შემთხვევაში, რეგისტრში შეიცვლება მხოლოდ გამოყენებული ბიტები, ხოლო დანარჩენი დარჩება ხელუხლებელი. ეს გარანტიას იძლევა, რომ მიმნიჭებელი არ დაარღვევს სხვა კომპონენტის გამართულ მუშაობას, თუ ეს კომპონენტი ინახავს ბიტებს იგივე რეგისტრში. მაგრამ ამ შემთხვევაში წარმოქმნილი კოდი უფრო გრძელი იქნება.

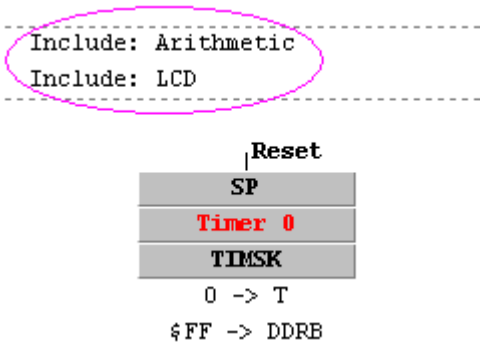
ელემენტი TEXT - ადგილობრივი ტექსტური რედაქტორის სტრიქონი

ეს ელემენტი არის ტექსტური სტრიქონი, რომელიც იწყება ალგორითმის ველის მარცხენა მხარეს. ასეთი სტრიქონების ერთობლიობა წარმოადგენს ლოკალურ ტექსტურ რედაქტორს, რომელიც შემოსაზღვრულია წერტილოვანი ხაზებით. მასში მუშაობის წესები სხვა ტექსტური რედაქტორების ანალოგიურია.

სტრიქონები გამოიყენება მათში კომპილატორისთვის(ამწყობისთვის) დირექტივებისა (მითითებების) და კომენტარების ჩაწერისთვის. ელემენტის დამატებისთვის გამოიყენეთ მენიუს ელემენტი “Elements\Text”, ან დააჭირეთ ხელსაწყოთა პანელზე მყოფ ღილაკს  , ან კლავიატურის ღილაკთა კომბინაციას “Alt+T”.

კომენტარი უნდა დაიწყოს ორი დახრილი ხაზით: “//”.

ქვემოთ მოცემულ მაგალითში TEXT ტიპის ელემენტები ოვალებშია მოთავსებული:



პროექტის ფაილები

რედაქტორში შექმნილი ალგრითმის გვერდი დისკზე შეინახება ფორმატში ***.alp** პროექტის მთავარი (პირველი) და ***.alg** დანარჩენი გვერდებისთვის. რედაქტორში ფაილის ჩატვირთვისას იქმნება მისი ასლი გადართოებით ***.~al**, რომელიც საჭიროებისამებრ შეგიძლიათ გამოიყენოთ ფაილის აღდგენისთვის.

მიმდინარე პროექტის გარემო შეინახება პროექტის სახელის მქონე **.ini** ფაილში. მასში შეინახება ყველა ალგორითმის ჩამონათვალი, რომელიც გახსნილი იყო პროგრამის დახურვის მომენტში, მთავარი და სიმულატორის ფანჯრების მდებარეობა და მდგომარეობა.

წარმატებული დაკომპილების შემდეგ დისკზე იქმნება მეხსიერების შემცველი ფაილი, რომელიც პროექტის სახელს ატარებს და EEPROM შიგთავსის შემცველი ფაილი თავსართით "**EE_**". ფაილები შეიძლება შექმნილი იყოს ბინარულ ფორმატში ***.bin**, ტექსტურ ფორმატში "Generic" გაფართოებით ***.gen** ან ფორმატში "Intel HEX" გაფართოებით ***.hex.**

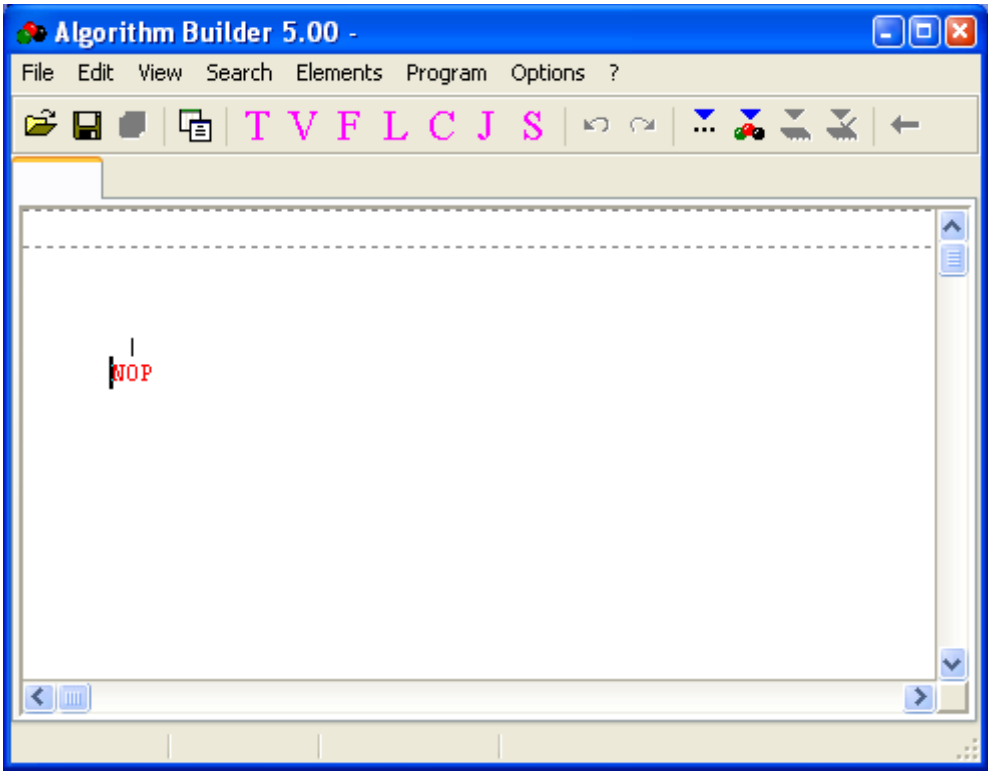
ფორმატის შერჩევა ხდება მენიუში "Options\Environment Options..."

გარემოს აღწერა

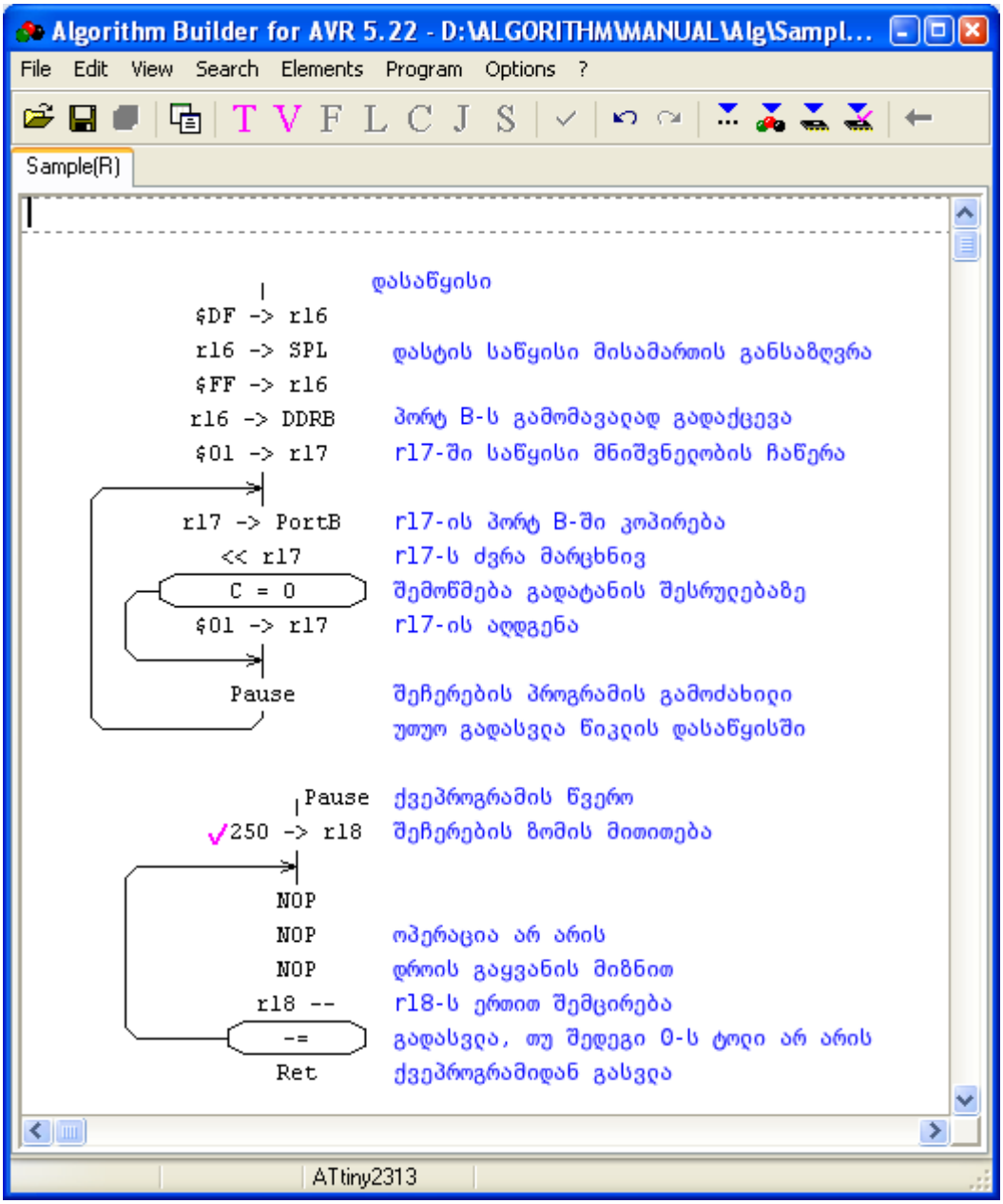
მარტივი ალგორითმის მაგალითი

თუ ამჟამად გახსნილი გაქვთ რაიმე ალგორითმი, დახურეთ ის. ამისთვის გამოიყენეთ მენიუს ელემენტი “File\Close Project”.

ახალი პროექტის შექმნისთვის მენიუდან აირჩიეთ “File\New”. ამასთან გაჩნდება ახალი ჩანართი და სამუშაო არეზე შეიქმნება ელემენტი TEXT, VERTEX და მასზე მიმაგრებული FIELD ცარიელი ოპერატორით “NOP”:



შეიყვანეთ ქვემოთ მოცემული მარტივი ალგორითმის მაგალითი:



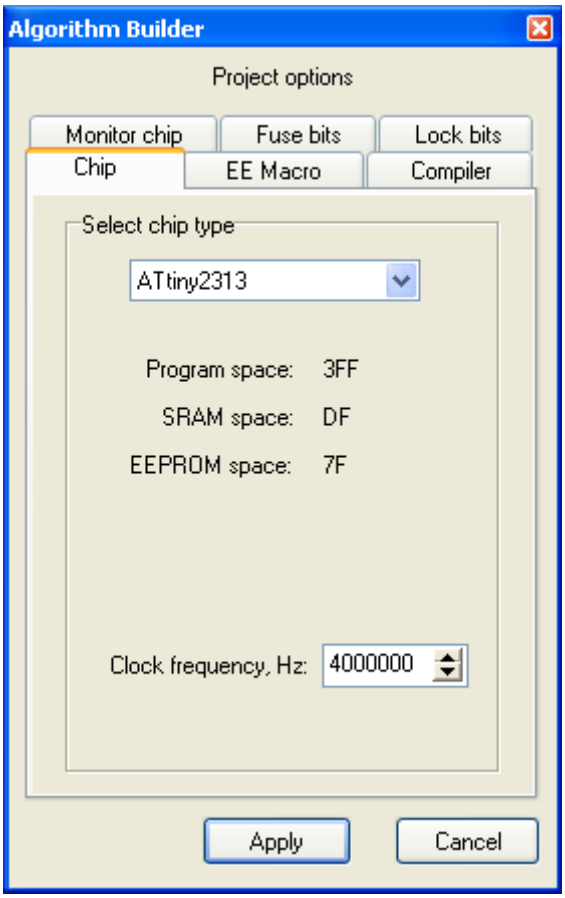
ამ ალგირთმს გამოყავს ლოგიკური ერთიანები B პორტის ბიტებში. ალგორითმი შეიცავს მიკროკონტროლერის ელემენტარულ მითითებებს. ეს მაგალითი მოთავსებულია დართულ საქაღალდეში “Sample0(R)”, მაგრამ მიზანშეწონილია ხელით შეიყვანოთ.

კოპირების ოპერაციისთვის ხშირად გამოიყენება სიმბოლოები “->”. შეყვანის მოხერხებულობისთვის, მათი აკრეფა შესაძლებელია ღილაკით “~” (მდებარეობს Esc-ის ქვეშ). კიდევ ერთი FIELD ელემენტის დამატებისთვის შეიძლება უბრალოდ დააჭიროთ ღილაკს “Enter”, ახალი ჯდე – ღილაკები “Alt+L”, პირობითი გადასვლა - “Alt+C”, უთუო გადასვლა - “Alt+J”. ასევე, ამ ელემენტების დამატება შესაძლებელია ხელსაწყოთა პანელზე მყოფი ღილაკებით.

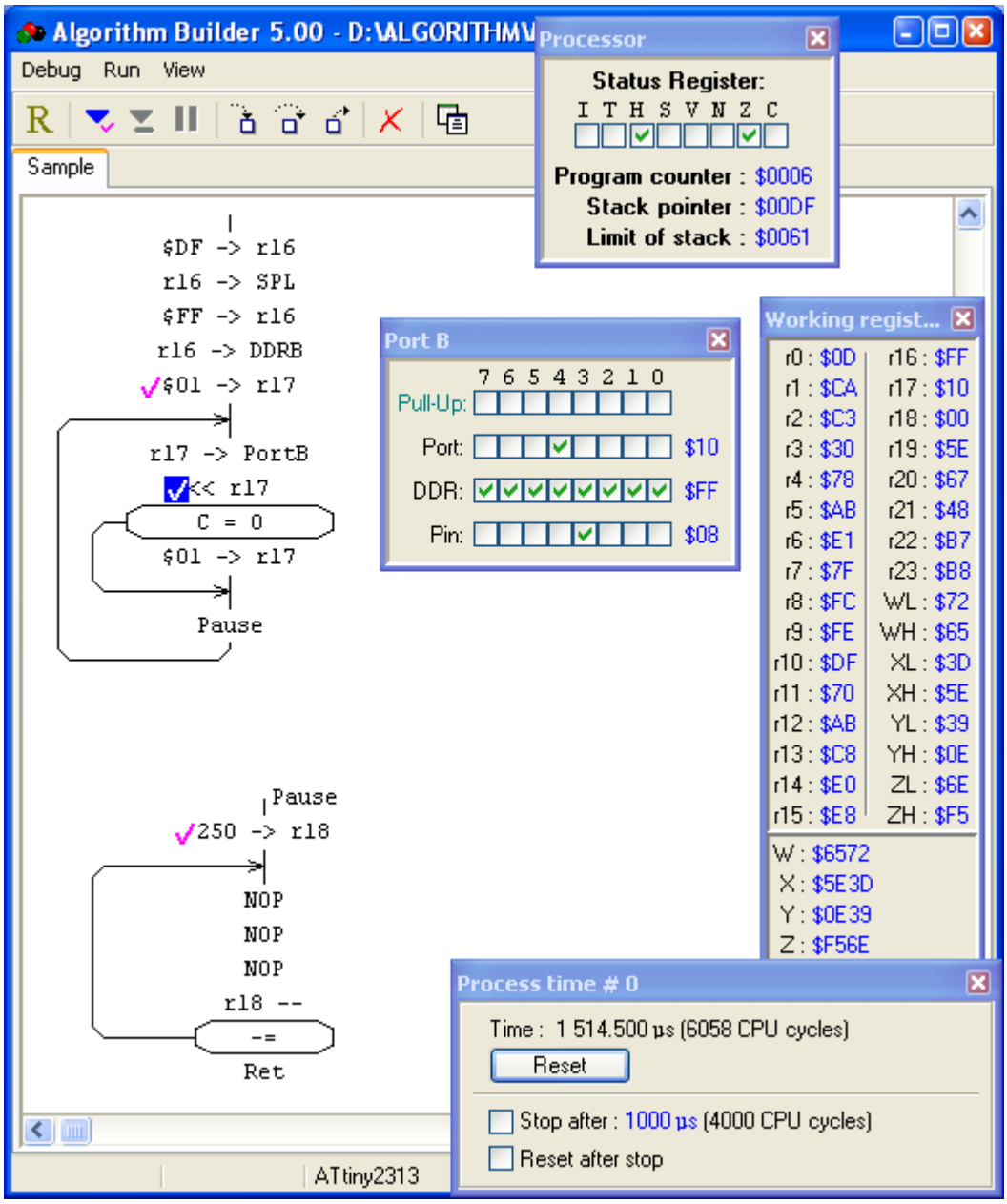
ალგორითმში ახალი ბლოკის დამატებისთვის მოათავსეთ რედაქტორის ფოკუსი წინა ბლოკის ბოლო ელემენტზე და დააჭირეთ კომბინაციას “Alt+V”.

იისფერი თოლია ✓ ოპერატორთან “250->r18” არის სიმულატორის შეჩერების წერტილი. მისი დამატება ან ამოღებისთვის მოსდეთ რედაქტორის ფოკუსი ოპერატორზე და დააჭირეთ ღილაკს “F5”.

დაკომპილების დაწყებამდე საჭიროა მიუთითოთ მიკროკონტროლერის ტიპი. ამისთვის გამოიყენეთ მენიუს ელემენტი “Options\Project options...”:



ალგორითმის სიმულატორში შესრულებისთვის, დააჭირეთ ღილაკს “F9” ან ხელსაწყოთა პანელზე მყოფ ღილაკს . ამასთან, ჯერ მოხდება დაკომპილება და, თუ არ მოიძებნა შეცდომები, გაიშვება სიმულატორი. მიკროკონტროლერის სასურველი კომპონენტების ფანჯრების გახსნა ხდება “View...” მენიუს გამოყენებით. შეყვანილ ალგორითმში მიმდინარე პროცესებს შეგიძლიათ ადევნოთ თვალი ფანჯრებში “Processor”, “PortB” და “Working Registers”.



ოპერატორის გვერდზე მყოფი ხატულა ☑ აჩვენებს პროგრამული მთვლელის მიმდინარე მდებარეობას. ხატულა მოთავსებულია იმ ოპერატორის წინ, რომელიც უნდა შესრულდეს შემდეგ ნაბიჯში. გაშვებისთანავე, ის მოთავსებულია პროგრამის პირველ ოპერატორთან, რომელიც მდებარეობს მისამართზე $0000.

სიმულატორის გაშვებისას, მუშა რეგისტრები და SRAM ყოველთვის ივსება შემთხვევითი სიდიდეებით, რადგან ნამდვილ მიკროკონტროლერში კვების მიწოდების შემდეგ შეუძლებელია მათი მნიშვნელობის გამოცნობა.

ალგორითმის ქვეპროგრამებთან ერთად ნაბიჯ-ნაბიჯ შესრულება ხდება კლავიატურის ღილაკით “F7” ან ხელსაწყოთა პანელზე მყოფი ღილაკით . ნაბიჯ-ნაბიჯ შესრულება ქვეპროგრამების შეუსრულებლად ხდება ღილაკით “F8” ან , ალგორითმის შესრულება ქვეპროგრამიდან გასვლამდე – ღილაკით “F6” ან .

ალგორითმის შეუწყვეტლად შესრულებისთვის შეჩერების წერტილამდე (☑) გამოიყენება ღილაკი “F9” ან ღილაკი , ხოლო მონიშნულ ელემენტამდე – ღილაკი “F4” ან . ამასთან ალგორითმის შესრულების შეჩერება ღიკალკი “F2” ან .

მოცემული ალგორითმის შესრულების პროცესში შესაძლებელია დავინახოთ B პორტის გასავალზე ლოგიკური ერთიანების გაჩენა.

სურვილისამებრ შეიძლება ამ ალგორითმის ნამდვილ კრისტალში ჩაწერა და მისი

მუშაობისთვის ოსცილოგრაფით თვალყურის დევნება.

მიკროკონტროლერის ოპერატორები

AVR ბრძანებათა სისტემა შეიცავს 130-ზე მეტ მარტივ ოპერაციას. ქვემოთ მოყვანილ ცხრილებში ჩამოთვლილია ყველა არსებული ქმედება.

კოპირების და არითმეტიკა-ლოგიკური გარდაქმნების ქმედებები

მათი ჩაწერისთვის გამოიყენება მხოლოდ ელემენტები FIELD.

შაბლონი	მოქმედება	მაგალითი	ანალოგი
R-> R	ერთი მოქმედი რეგისტრის ასლის მეორეში ჩაწერა	R0 -> R1	MOV R,R
# -> R	# მუდმივის რეგისტრში ჩაწერა	24 -> r16	LDI R,K
[X] -> R	SRAM-იდან რეგისტრში ირიბად ჩაწერა X-თ	[X] -> R2	LD R,X
[X++] -> R	SRAM-იდან რეგისტრში ირიბად ჩაწერა X-ს პოსტ-ინკრემენტით	[X++] -> R3	LD R,X+
[--X] -> R	SRAM-იდან რეგისტრში ირიბად ჩაწერა X-ს პრე-დეკრემენტით	[--X] -> R4	LD R,-X
[Y] -> R	SRAM-იდან რეგისტრში ირიბად ჩაწერა Y-თ	[Y] -> R5	LD R,Y
[Y++] -> R	SRAM-იდან რეგისტრში ირიბად ჩაწერა Y-ს პოსტ-ინკრემენტით	[Y++] -> R6	LD R,Y+
[--Y] -> R	SRAM-იდან რეგისტრში ირიბად ჩაწერა Y-ს პრე-დეკრემენტით	[--Y] -> R7	LD R,-Y
[Y+#] -> R	SRAM-იდან რეგისტრში ირიბად ჩაწერა Y-დან # წანაცვლებით	[Y+2] -> R8	LDD R,Y+q
[Z] -> R	SRAM-იდან რეგისტრში ირიბად ჩაწერა Z-თ	[Z] -> R9	LD R,Z
[Z++] -> R	SRAM-იდან რეგისტრში ირიბად ჩაწერა Z-ს პოსტ-ინკრემენტით	[Z++] -> R10	LD R,Z+
[--Z] -> R	SRAM-იდან რეგისტრში ირიბად ჩაწერა Z-ს პრე-დეკრემენტით	[--Z] -> R11	LD R,-Z
[Z+#] -> R	SRAM-იდან რეგისტრში ირიბად ჩაწერა Z-დან # წანაცვლებით	[Z+1] -> R12	LDD R,Z+q
[#] -> R	SRAM-იდან რეგისტრში ჩაწერა უშუალოდ # მისამართიდან	[$60] -> R14	LDS R,k
R -> [X]	რეგისტრიდან SRAM-ში ირიბად ჩაწერა X-თ	R15 -> [X]	ST X,R
R -> [X++]	რეგისტრიდან SRAM-ში ირიბად ჩაწერა X-ს პოსტ-ინკრემენტით	R16 -> [X++]	ST X+,R
R -> [--X]	რეგისტრიდან SRAM-ში ირიბად ჩაწერა X-ს პრე-დეკრემენტით	R17 -> [--X]	ST -X,R
R -> [Y]	რეგისტრიდან SRAM-ში ირიბად ჩაწერა Y-თ	R18 -> [Y]	ST Y,R
R -> [Y++]	რეგისტრიდან SRAM-ში ირიბად ჩაწერა Y-ს პოსტ-ინკრემენტით	R19 -> [Y++]	ST Y+,R
R -> [--Y]	რეგისტრიდან SRAM-ში ირიბად ჩაწერა Y-ს პრე-დეკრემენტით	R20 -> [--Y]	ST –Y,R

R -> [Y+#]	რეგისტრიდან SRAM-ში ირიბად ჩაწერა Y-დან # წანაცვლებით	R21 -> [Y+2]	STD Y+q,R
R -> [Z]	რეგისტრიდან SRAM-ში ირიბად ჩაწერა Z-ით	R22 -> [Z]	ST Z,R
R -> [Z++]	რეგისტრიდან SRAM-ში ირიბად ჩაწერა Z-ს პოსტ-ინკრემენტით	R23 -> [Z++]	SR Z+,R
R -> [--Z]	რეგისტრიდან SRAM-ში ირიბად ჩაწერა Z-ს პრე-დეკრემენტით	R24 -> [--Z]	ST -Z,R
R -> [Z+#]	რეგისტრიდან SRAM-ში ირიბად ჩაწერა Z-დან # წანაცვლებით	R25 -> [Z+3]	STD Z+q,R
R -> [#]	რეგისტრიდან SRAM-ში ჩაწერა უშუალოდ # მისამართზე	R26 -> [$200]	STS k,R
LPM LPM[Z]	პროგრამის მეხსიერებიდან ჩატვირთვა Z-ით r0-ში		LPM
LPM -> R LPM[Z] -> R	პროგრამის მეხსიერებიდან ჩატვირთვა Z-ით სამუშაო რეგისტრში		LPM R,Z
LPM[Z++] -> R	პროგრამის მეხსიერებიდან ჩატვირთვა Z-ის პოსტ-ინკრემენტით 2 სამუშაო რეგისტრში	LPM[Z++]-> r5	LPM R,Z+
SPM	პროგრამის მეხსიერებაში ჩაწერა		SPM
P -> R	I/O რეგისტრის ასღი მოთავსება სამუშაო რეგისტრში	P$19 -> WH	IN R,P
R -> P	სამუშაო რეგისტრის ასღის მოთავსება I/O რეგისტრში	XL -> PortB	OUT P,R
R->	სამუშაო რეგისტრის ასღის დასტაში მოთავსება	r0->	PUSH R
->R	სტეკიდან სამუშაო რეგისტრში ასღის მოთავსება	->r1	POP R
NOP	ქმედება არ არის		NOP
R + R	მარჯვენა რეგისტრის მიმატება მარცხენასთან	r0 + r1	ADD R,R
R + R +	მარჯვენა რეგისტრის მიმატება მარცხენასთან (გადატანით)	R0 + R5 +	ADC R,R
RR + #	ორმაგი სამუშაო რეგისტრის (W, X, Y, Z) გაზრდა #-ით	W + 32	ADIW RWl,K
R - R	მარცხენა რეგისტრიდან მარჯვენას გამოკლება	r4 - r5	SUB R,Rr
R - #	სამუშაო რეგისტრიდან მუდმივის გამოკლება	XL - 2	SUBI R,K
R + #	რეგისტრზე მუდმივის მიმატება	R17 + 12	SUBI R,256-K
R - R -	მარცხენა სამუშაო რეგისტრიდან მარჯვენის გამოკლება გადატანით	r5 - r6 -	SBC R,R
R - # -	მარცხენა სამუშაო რეგისტრიდან მუდმივის გამოკლება გადატანით	ZL - $70 -	SBCI R,K
RR - #	ორმაგი სამუშაო რეგისტრიდან (W, X, Y, Z) # მუდმივის გამოკლება	X - $2E	SBIW RWl,K
R & R	ბიტური ღოგიკური ქმედება **'და'** ორი სამუშაო რეგისტრისთვის	r7 & r8	AND R,R

R & #	ბიტური ლოგიკური ქმედება **'და'** სამუშაო რეგისტრისა და მუდმივისთის	r17 & #b00111101	ANDI R,K
R &| #	ბიტური ლოგიკური ქმედება **'და'** სამუშაო რეგისტრისა და შებრუნებული მუდმივისთის	r18 &| #b00010011	CBR R,K
R ! R	ბიტური ლოგიკური ქმედება **'ან'** ორი სამუშაო რეგისტრისთვის	R9 ! r10	OR R,R
R ! #	ბიტური ლოგიკური ქმედება **'ან'** სამუშაო რეგისტრისა და მუდმივისთვის	R18 ! #o147	ORI R,K
R ^ R	ბიტური ლოგიკური ქმედება **'გამომრიცხვავი ან'** ორი სამუშაო რეგისტრისთვის	R11 ^ r12	EOR R,R
|R	სამუშაო რეგისტრის ბიტური შებრუნება	-r8-	COM R
-R	სამუშაო რეგისტრის არითმეტიკული შებრუნება	-RB	NEG R
R++	სამუშაო რეგისტრის ერთით გაზრდა	Count++	INC R
R--	სამუშაო რეგისტრის ერთით შემცირება	Idx--	DEC R
R?	სამუშაო რეგისტრის გასინჯვა (R & R)	R16?	TST R
^R	სამუშაო რეგისტრის გაწმენდა (R ^ R)	R8	CLR R
R * R	სამუშაო რეგისტრების გადამრავლება	R6 * R17	MUL R,R
±R * ±R	სამუშაო რეგისტრების გადამრავლება ნიშნების გათვალისწინებით	±r16 * ±r20	MULS R,R
±R * R R * ±R	სამუშაო რეგისტრების გადამრავლება ერთ-ერთის ნიშნის გათვალისწინებით	±r18 * r22 r22 * ±r18	MULSU R,R
<<(R * R)	ორი სამუშაო რეგისტრის წილადური ნამრავლი	<<(R6 * R17)	FMUL R,R
<<(±R * ±R)	ორი სამუშაო რეგისტრის წილადური ნამრავლი ნიშნების გათვალისწინებით	<<(±r16 * ±r20)	FMULS R,R
<<(±R * R) <<(R * ±R)	ორი სამუშაო რეგისტრის წილადური ნამრავლი ერთ-ერთის ნიშნის გათვალისწინებით	<<(±r18 * r22) <<(r22 * ±r18)	FMULSU R,R
1 -> P.#	1-ს ჩაწერა I/O რეგისტრის ბიტში	1 -> DDB2	SBI P,b
0 -> P.#	0-ს ჩაწერა I/O რეგისტრის ბიტში	0 -> PortD.7	CBI P,b
<<R	სამუშაო რეგისტრის ლოგიკურად დაძვრა მარცხნივ	<<r18	LSL R
R>>	სამუშაო რეგისტრის ლოგიკურად დაძვრა მარჯვნივ	R19>>	LSR R
<<R<	სამუშაო რეგისტრის დაძვრა მარცხნივ ლოგიკური გადატანით	<<r20<	ROL R
>R>>	სამუშაო რეგისტრის დაძვრა მარჯვნივ ლოგიკური გადატანით	>r21>>	ROR R
±R>>	სამუშაო რეგისტრის არითმეტიკული დაძვრა მარჯვნივ	±r22>>	ASR R
>>R<<	სამუშაო რეგისტრების “ტეტრადების” ადგილების გაცვლა	>>r23<<	SWAP R
R.#->	სამუშაო რეგისტრის მე-# ბიტის ჩაწერა T-ში (SREG)	RA.6->	BST R,b

->R.#	T-ს (SREG) ჩაწერა სამუშაო რეგისტრის მე-# ბიტში	->r0.4	BLD R,b
# -> R.#	#-ს (0 ან 1) ჩაწერა სამუშაო რეგისტრის მე-# ბიტში	0 -> r18.4 1 -> r20.0	ANDI R,256-K ORI R,K
RR -> RR	ორმაგი სამუშაო რეგისტრის ასღის ჩაწერა სხვა ორმაგ სამუშაო რეგისტრში	Y -> X	MOVW R,R
1 -> C	1-ს ჩაწერა C ბიტში		SEC
0 -> C	0-ს ჩაწერა C ბიტში		CLC
1 -> N	1-ს ჩაწერა N ბიტში		SEN
0 -> N	0-ს ჩაწერა N ბიტში		CLN
1 -> Z 1 -> .Z	1-ს ჩაწერა Z ბიტში		SEZ
0 -> Z 0 -> .Z	0-ს ჩაწერა Z ბიტში		CLZ
1 -> I	1-ს ჩაწერა I ბიტში		SEI
0 -> I	0-ს ჩაწერა I ბიტში		CLI
1 -> S	1-ს ჩაწერა S ბიტში		SES
0 -> S	0-ს ჩაწერა S ბიტში		CLS
1 -> V	1-ს ჩაწერა V ბიტში		SEV
0 -> V	0-ს ჩაწერა V ბიტში		CLV
1 -> T ; 1->	1-ს ჩაწერა T ბიტში		SET
0 -> T ; 0->	0-ს ჩაწერა T ბიტში		CLT
1 -> H	1-ს ჩაწერა H ბიტში		SEH
0 -> H	0-ს ჩაწერა H ბიტში		CLH
SLEEP	კრისტაღის მუშაობის შეჩერება		SLEEP
WDR	დამცავი წამზომის ჩამოყრა		WDR
JMP #	დაშორებუღი უთუო გადასვღა	JMP LabelName	JMP k
#/	ქვეპროგრამის შორეუღი გამოძახება	LabelName/	CALL k
JMP[Z]	ირიბი უთუო გადასვღა Z-ით		IJMP
#	ქვეპროგრამის ფარდობითი გამოძახება	LabelName	RCALL k
CALL[Z]	ქვეპროგრამის ირიბად გამოძახება Z-ით	Call[Z]	ICALL
RET	ქვეპროგრამიდან დაბრუნება	RET	RET
RETI	ქვეპროგრამიდან დაბრუნება და I ბიტში 1-ს ჩაწერა	RETI	RETI
R = R	ორი სამუშაო რეგისტრის შედარება	R2 = R5	CP R,R
R = R =	ორი სამუშაო რეგისტრის შედარება გადატანით	R6 = R5 =	CPC R,R
R = #	სამუშაო რეგისტრისა და მუდმივის შედარება	YL = 47	CPI R,K

გამრავლების ყველა ქმედებაში შედეგი ჩაიწერება ორმაგ რეგისტრში R0, R1.

ასევე, ყურადღება უნდა მიექცეს იმას, რომ ნებისმიერი ოპერაციები რეგისტრისა და მუდმივას შორის (მაგ. “# -> R” ან “R & #”) შესაძლებელია მხოლოდ r16...r31-თვის, ხოლო მოქმედება I/O რეგისტრების ბიტებთან (მაგ. “1 -> PortB.4”) – მხოლოდ p0...p31-თვის.

პირობითი გადასვლის ოპერატორები

ამ ოპერატორების გამოყენება ხდება ელემენტებში CONDITION, რომელშიც შესაძლებელია შემდეგი პირობების ჩაწერა:

პირობის შაბლონი	კომენტარები	ანალოგი
=	შედეგი ნულის ტოლია (Z = 1)	BREQ k
-=	შედეგი ნულის ტოლი არ არის (Z = 0)	BRNE k
>=	მეტი ან ტოლი	BRSH k
<	ნაკლები	BRLO k
-	უარყოფითი შედეგი	BRMI k
+	დადებითი შედეგი	BRPL k
±>=	მეტი ან ტოლი (ნიშნის გათვალისწინებით)	BRGE k
±<	ნაკლები (ნიშნის გათვალისწინებით)	BRLT k
C = 1		BRCS k
C = 0		BRCC k
H = 1		BRHS k
H = 0		BTHC k
T = 1		BRTS k
T = 0		BRTC k
V = 1		BRVS k
V = 0		BRVC k
I = 1		BRIE k
I = 0		BRID k

CONDITION ელემენტის ვექტორი უნდა მთავრდებოდეს ან ჭდეზე ან წვეროზე, ან უნდა შეიცავდეს სასურველი ჭდის მისამართს.

შემდეგი ოპერატორის პირობითი გამოტოვების ოპერატორები

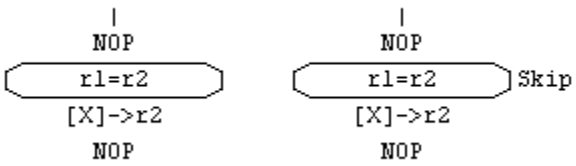
ამ შემთხვევაში CONDITION ელემენტის ვექტორის ან არ არის, ან შეიცავს რეზერვებულ სიტყვას “Skip”

პირობის შაბლონი	კომენტარები	მაგალითი	ანალოგი
R = R	ორი სამუშაო რეგისტრის ტოლობა	r0 = r1	CPSE R,R
R.# = 0	სამუშაო რეგისტრის მე-# ბიტი = 0	XH.2 = 0	SBRC R, b
R.# = 1	სამუშაო რეგისტრის მე-# ბიტი = 1	r14.3 = 1	SBRS R, b
P.# = 0	I/O რეგისტრის მე-# ბიტი = 0	PinC.6 = 0	SBIC P, b
P.# = 1	I/O რეგისტრის მე-# ბიტი = 1	EERE = 1	SBIS P, b

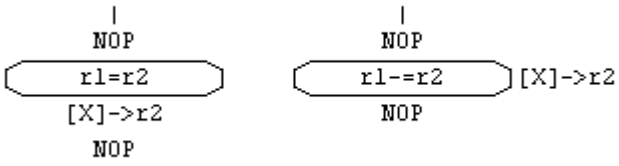
აქ შესაძლებელია CONDITION ელემენტის ორი კონსტრუქცია:

1. ელემენტის ვექტორი არ არის, ხოლო ვექტორის სახელი ან არ არის, ან შეიცავს რეზერვებული სიტყვას “Skip”. ქვემოთ მოყვანილ მაგალითებში:

“[X] -> r2” გამოიტოვება, თუ შესრულდება პირობა “r1 = r2”.



2. ჩაიწერება ინვერსული პირობა, ვექტორი არ არის, ხოლო გამოსატოვებელი ქმედება ჩაიწერება ვექტორის სახელის ადგილას. ამ შემთხვევაში იგულისხმება: ბრძანება შესრულდება მხოლოდ იმ შემთხვევაში, თუ შესრულდება პირობა. ქვემოთმოყვანილი მაგალითები ერთნაირია:



კომენტარები ოპერატორებთან დაკავშირებით

არჩეული სახეობის მიკროკონტროლერის შაბლონების შემცველი ფანჯრის გახსნა შეიძლება მენიუდან “View\Templates...” და საჭიროების შემთხვევაში, სასურველზე დაწკაპვითა და “Insert” ღილაკის დაჭერით ჩატვირთოთ ის ჩასწორებადი ობიექტის სახით.

უნდა გაითვალისწინოთ, რომ მიკროკონტროლერების ზოგიერთ სახეობაში შესაძლოა არ აღმოჩნდეს ოპერატორების სრული კრებული.

ოპერატორების შაბლონში სიმბოლო "#" აღნიშნავს მუდმივას. ისინი შესაძლოა იყვნენ როგორც დადებითი, ისე უარყოფითი. მაგალითად ოპერატორები:

```
-2 -> r16 და $FE -> r16
```

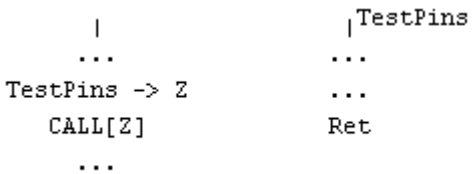
ერთანირად იქნება დაკომპილებული.

ასევე, ყურადღება ინდა მიექცეს უთუო გადასვლებისა და ქვეპროგრამების გამოყენების თავისებურებებს:

- უთუო გადასვლა მიიღწევა მხოლოდ და მხოლოდ ელემენტით JMP Vector. ამასთან, თუ გადასვლის მანძილი არ აღემატება 2047 პუნქტს, კომპილატორის გამოიყენებს მოკლე ფარდობით გადასვლას ("RJMP"), წინააღმდეგ შემთხვევაში – გრძელს ("JMP k").
- ქვეპროგრამის გამოძახება მიიღწევა ჭდის(წვეროს) დასახელების ჩაწერით ან მუდმივის თუ მისი ელემენტი FIELD შეიცავს ქვეპროგრამის მისამართს. ამასთან, თუ გადასვლის მანძილი არ აღემატება 2047 პუნქტს, კომპილატორი გამოიყენებს მოკლე ფარდობით გამოძახებას ("RCALL"), წინააღმდეგ შემთხვევაში – გრძელს ("CALL k").

  მაგალითად:

- ირიბი გადასვლა (ანალოგი – "IJMP") და ქვეპროგრამის გამოძახება (ანალოგი – "ICALL") მიიღება შესაბამისი ჩაწერებით "JMP[Z]" და "CALL[Z]" ელემენტში FIELD პარამეტრების გარეშე. მაგალითად:



მუდმივების წარდგენა უშუალი სახით

შაბლონებში, სიმბოლო "#" აღნიშნავს მუდმივას, რომლის წარდგენა ხდება უშუალო სახით.

Algorithm Builder-ში შესაძლებელია მუდმივების წარდგენა ჩვეულებრივი – ათობითი სახით, თექსვმეტობითი, რვაობითი, ორობითი, სიმბოლური სახით და *ათწილადების ფორმატში[??]* მოტივტივე ნიშნით.

ათობითი სახის მუდმივა ჩაიწერება სიმბოლოებით "0...9", მაგ. "35", "24000".

თექსვმეტობითი სახის მუდმივის ჩაწერა იწყება სიმბოლოებით "$", "#h" ან "#H", შემდეგ კი გამოიყენება სიმბოლოები "0...9" და "A...F" ან "a...f". მაგალითად: "#h0f", "#He5", "$23E7".

რვაობითი სახის მუდმივის ჩაწერა იწყება სიმბოლოებით "#o" ან "#O", შემდეგ გამოიყენება სიმბოლოები "0...7", მაგალითად: "#o107", "#O2071".

ორობითი სახის მუდმივა იწყება სიმბოლოებით "#b" ან "#B", შემდეგ კი გამოიყენება სიმბოლოები "0" და "1". მაგალითად: "#b01101101", "#B10011110".

მუდმივის **სიმბოლური** სახით წარდგენის დროს გამოიყენება ბრჭყალები. ამ შემთხვევაში მუდმივის მნიშვნელობა იქნება მასში ჩაწერილი სიმბოლოს ANSI კოდი. მაგალითად, მუდმივების ასეთი წარდგენა: "0" და $30 იქნება ექვივალენტური.

თუ წარდგენა შეიცავს რამდენიმე სიმბოლოს, მაშინ ასეთი მუდმივა იქნება მრავალბაიტიანი. მაგალითად, "12" იქნება $3231 მუდმივის ტოლი.

რესურსების განაწილება და სახელების გამოცხადება

დაპროგრამება Algorithm Builder-ში შესაძლებელია სტანდარტული სამუშაო და I/O რეგისტრების სახელების გამოყენებით, ხოლო SRAM და EEPROM მეხსიერების უჯრედების მიმართვა ხდება უშუალოდ ფიზიკური მისამართით. ასეთი დაპროგრამება მოუხერხებელია.

მიკროკონტროლერის რესურსების განაწილება და სახელების გამოცხადება ხდება განსაკუთრებულ ელცხრილში. რათა გადართოთ ელცხრილის ხილვადობა, გამოიყენეთ მენიუს ელემენტი "View\Toggle Algorithm/Data table" ან დააჭირეთ ხელსაწყოთა პანელზე მყოფ ღილაკს  , ან კლავიატურის ღილაკს "F12".

სახელების გამოცხადებისთვის შეიძლება გამოიყენოთ სიმბოლოები: "A...Z", "a...z", ციფრები "0...9" და ქვედა დეფისი "_". ამასთან, პირველი სიმბოლო არ შეიძლეაბა იყოს ციფრი. კომპილატორი არ განანსხვავებს ზედა და ქვედა რეგისტრის სიმბოლოებს, ამიტომ სახელები როგორიცაა: "GenerateNoise", "GENERATENOISE" და "generatenoise" დამუშავდება როგორც ერთნაირი.

რესურსების განაწილებისთვის და სახელების გამოცხადებისთვის ელცხრილში გათვალისწინებულია ექვსი განყოფილება:

- სამუშაო რეგისტრების სახელების გამოცხადება;
- შეტანა/გამოტანის (I/O) რეგისტრების სახელების გამოცხადება;
- რეგისტრების ბიტების სახელების გამოცხადება;
- SRAM ცვლადების სახელების გამოცხადება;
- EEPROM ცვლადების სახელების გამოცხადება;
- მუდმივების სახელების გამოცხადება.

ნებისმიერი განყოფილების გამოყენების საჭიროებას განსაზღვრავს თვით პროგრამისტი.

ცვლადთა ფორმატები

ფორმატების განსაზღვრა ხდება სამუშაო რეგისტრების, შეტანა/გამოტანის რეგისტრების, SRAM და EEPROM ცვლადების, მუდმივთა მასივებისთვის (რომლებიც განთავსდება პროგრამის მეხსიერებაში) სახელების მინიჭების დროს.

Algorithm Builder-ში გათვალისწინებულია შემდეგი ფორმატები:

ფორმატი	საკვანძო სიტყვა
1 ბაიტი	“Int8” ან “Byte”
2 ბაიტი	“Int16” ან “Word”
3 ბაიტი	“Int24”
4 ბაიტი	“Int32” ან “DWord”
5 ბაიტი	“Int40”
6 ბაიტი	“Int48”
7 ბაიტი	“Int56”
8 ბაიტი	“Int64” ან “QWord”

ნაგულისხმევად ყოველთვის გამოიყენება ერთბაიტიანი ფორმატი

მუდმივების გამოცხადების განყოფილება

დაპროგრამებისას შესაძლებელია გამოიყენოთ მუდმივები მათი უშუალო სახით. იმ შემთხვევაში, თუ ერთი და იგივე მუდმივა გამოიყენება ალგორითმის რამდენიმე ადგილას, მაშინ უფრო მოსახერხებელი იქნება თუ მას მიანიჭებთ სახელს და გამოიყენებთ ოპერატორებში. ამასთან, თუ საჭირო გახდება მისი მნიშვნელობის შეცვლა, საკმარისი იქნება შეცვალოთ ის მხოლოდ გამოცხადების ადგილას. ამის გარდა, სახელების გამოყენება საგრძნობლად ამარტივებს ალგორითმის აღქმას.

განყოფილების დასახელება: `Constants:`

გათვალისწინებულია შემდეგი ველები:

- `Name` - გამოსაცხადებელი სახელი;
- `Value` - სიდიდის განმსაზღვრელი ალგებრული გამოსახულება.

განყოფილების შევსების მაგალითი:

Constants:

Name	Value	Commentary
Weight_Index	7	მუდმივა Weight_Index სიდიდით 7
Weight_Min	10	
Weight_Typ	75	
Weight_Max	Weight_Min+100	
InputCode	$DC5E	

მუდმივიდან ბაიტის არჩევისთვის, საჭიროა მიაწეროთ "`.#`", სადაც `#` - ბაიტის ნომერია, დაწყებული 0-თ. მაგალითად გამოსახულება:

"`InputCode.1 -> r16`"

ჩაწერს `r16`-ში მუდმივას `$DC.`

ნაგულისხმევად, გარემოში გამოცხადებულია შემდეგი სისტემური მუდმივები:

- "`CPU_Clock_Frequency`" – ბირთვის მუშაობის სიხშირე ჰერცებში (Hz);
- "`IO_Org`" – საწყისი I/O რეგისტრი SRAM მისამართთა სივრცეში;

- "SRAM_Size" – SRAM-ის სიდიდე ბაიტებში;
- "SRAM_Org" – SRAM სივრცის დასაწყისი;
- "EEPROM_Size" – EEPROM-ის სიდიდე ბაიტებში;
- "Flash_Size" – Flash მეხსიერების სიდიდე 16-ბიტიან სიტყვებში;
- "Flash_Page_Size" – Flash მეხსიერების დაპროგრამების გვერდის სიდიდე 16-ბიტიან სიტყვებში;

სამუშაო რეგისტრთა სახელების გამოცხადების განყოფილება

განყოფილების დასახელება: `Working registers:`

გათვალისწინებულია შემდეგი ველები:

- Name - გამოსაცხადებელი სახელი;
- Index - *(არააუცილებელი)* მუდმივა, რომელიც განსაზღვრავს რეგისტრის ინდექსს. ნაგულისხმევად იღება ბოლო რეგისტრზე ერთით მეტი, ხოლო დაკომპილების დასაწყისში 0-ს ტოლია.
- Format - *(არააუცილებელი)* გამოცხადებული რეგისტრის ფორმატი. ნაგულისხმევად იღება ერთბაიტიანი ფორმატი. მრავალბაიტიანის გამოცხადებისას ავტომატურად ცხადდება მაში შესული ერთბაიტიანი რეგისტრების სახელები. ორბაიტიანი რეგისტრის დასახელებისას სახელებს დაერთვის სიმბოლოები "L" და "H", ხოლო სამ და ოთხბაიტიანებს – "0" და "1" და ა.შ. რიგის მიხედვით. მაგალითად, ორბაიტიანი ("Word") რეგისტრის დამატებისას სახელით "Counter", ავტომატურად შეიქმნება მისი შემადგენელი რეგისტრები სახელით "CounterL" და "CounterH". სამუსაო რეგისტრების მრავალბაიტიანი ფორმატები გამოიყენება მხოლოდ მაკრო-ოპერატორორებში.

გამოცხადებისას დასაშვებია ერთსა და იგივე რეგისტრს მიენიჭოს რამდენიმე სახელი.

ნაგულისხმევად, გარემოში მომქმედებს ერთბაიტიანი რეგისტრების სტანდარტული სახელები "r0...r31", ან "R0...R31", ასევე: "WL"="r24", "WH"="r25", "XL"="r26", "XH"="r27", "YL"="r28", "YH"="r29", "ZL"="r30", "ZH"="r31" და ორბაიტიანები ("Int16" ან "Word"): "W"=("r24,r25"), "X"=("r26,r27"), "Y"=("r28,r29") და "Z"=("r30,r31").

განყოფილების შევსების მაგალითი:

Working registers:

Name	Index	Format	Commentary
rr0	0	Int16	ორბაიტიანი სამუშაო რეგისტრი (r0, r1)
rr2		Int16	ორბაიტიანი სამუშაო რეგისტრი (r2, r3)
rrrr0	0	Int32	ოთხბაიტიანი სამუშაო რეგისტრი (r0. .r3)
rr16	16	Int16	ორბაიტიანი სამუშაო რეგისტრი (r16, r17)
ArCount			ერთბაიტიანი სამუშაო რეგისტრი (r18)

I\O რეგისტრთა სახელების გამოცხადების განყოფილება

შეტანა\გამოტანის რეგისტრთა განყოფილების დასახელება: I/O Registers

გათვალისწინებულია შემდეგი ველები:

- Name - გამოსაცხადებელი სახელი;
- I/O Register - სტანდარტული ან მანამდე გამოცხადებული შეტანა-გამოტანის რეგისტრის სახელი

გამოცხადებისას დასაშვებია ერთსა და იგივე რეგისტრს მიენიჭოს რამდენიმე სახელი.

ნაგულისხმევად, სისტემაში მომქმედებს სტანდარტული სახელები "p0...p63", როგორც

მიკროკონტროლერში არის განსაზღვრული "SPL", "SPH", "TCNT0", "EEDR" და ა.შ. გარდა ამისა, მაკრო-ოპერატორებში შესაძლებელია ორმაგი რეგისტრების გამოყენება, მაგალითად "ADC", "TCNT1".

შემთხვევათა უმეტესობაში ამ განყოფილების შევსების საჭიროება არ არის, რადგან ყველა გამოყენებული რეგისტრ ნაგულისხმევად აქვს სახელი.

განყოფილების შევსების მაგალითი:

I/O registers:		
Name	I/O register	Commentary
InputPort	PinA	PortA გამოცხადებული როგორც InputPort
OutputPort	p$22	p$22 გამოცხადებული როგორც OutputPort

ბიტების სახელების გამოცადება

განყოფილების დასახელება: Bits

ამ განყოფილებაში გათვალისწინებულია შემდეგი ველები:

- Name - გამოსაცხადებელი სახელი;
- Bit - არსებული ბიტის განმსაზღვრელი დასახელება ან ჩანაწერი

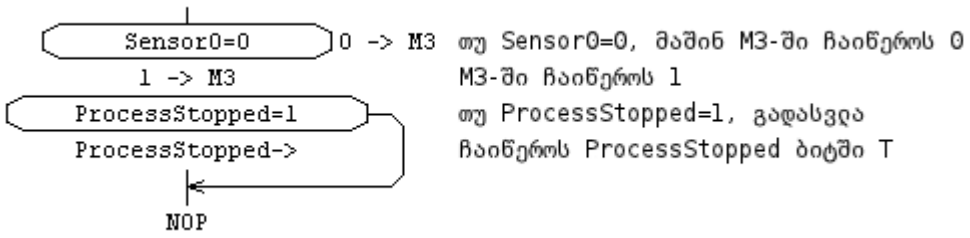
ნაგულისხმებად, გარემოში მოქმედებს ბიტების ტიპიური დასახელებები, რომლებიც განსაზღვრულია მიკროკონტროლერის ორიგინალურ აღწერაში, მაგალითად: "DDA7", "ACIC", "ADEN", "ICNC1" და ა.შ.

გამოცხადებული ბიტი შეიძლება ეკუთვნოდეს სამუშაო რეგისტრებს, შეტანა-გამოტანის რეგისტრებს, SRAM ან EEPROM ცვლადების რიცხვს.

განყოფილების შევსების მაგალითი:

Bits:		
Name	Bit	Commentary
Sensor0	PinA.5	PinA.5 აქვს სახე "Sensor0"
M3	PortD.4	PortD.4 აქვს სახე "M3"
ProcessStopped	r0.4	r0.4 აქვს სახე "ProcessorStopped"

ბიტების დასახელების გამოყენების მაგალითი:



SRAM ცვლადების გამოცხადების განყოფილება

განყოფილების დასახელება: SRAM

ამ განყოფილებაში გათვალისწინებულია შემდეგი ველები:

- Name - გამოსაცხადებელი ცვლადის სახელი (მეხსიერების უჯრედი);
- Address - *(არააუცილებელი)* მუდმივა, რომელიც განსაზღვრავს მისამართის მნიშვნელობას. ნაგულისხმებად – ბოლო განსაზღვრულის შემდეგი ან $60

დაკომპილების დაწყებისას;

- Format - *(არააუცილებელი)* ცვლადის ფორმატი. ნაგულისხმებად მიიღება ერთბაიტიანი ფორმატი. მრავალბაიტიანი ფორმატები გამოიყენება მაკრო-ოპერატორებში.
- Count - *(არააუცილებელი)* გადანახული(რეზერვებული) უჯრედების რაოდენობა. ნაგულისხმებად 1-ს ტოლია.

განყოფილების შევსების მაგალითი:

SRAM:

Name	Address	Format	Count	Commentary
EditIndex				ერთბაიტიანი ცვლადი
ShowMode		Word		ორბაიტიანი ცვლადი
Reg		Int24	4	ოთხი სამბაიტიანი ცვლადი
LCD_Page			16	16 ერთბაიტიანი უჯრედი
Phase	$100			ერთბაიტიანი ცვლადი მისამართით $100
XArray	@Phase+4		24	24 ერთბაიტიანი ცვლადი მისამართიდან $104

SRAM-ის უშუალო ადრესირების ოპერატორებში "[#]->R" და "R->[#]", "[#]"-ს მაგივრად შესაძლოა გამოყენებული იყოს ცვლადის სახელი.

ცვლადის სახელი პრეფიქსით “@” არის მუდმივა, რომელიც შეიცავს ფიზიკურ მისამართს SRAM-ში.

ქვემოთმოცემული მაგალითები გაივლის კომპილაციას ერთნაირი შედეგით:

```
[$100] -> r0 , Phase -> r0 , [@Phase] -> r0
```

გამოცხადებული მრავალბაიტიანი ცვლადები შეიძლება გამოყენებული იყოს მაკრო-ოპერატორებში, რომლებზეც ვისაუბრებთ შემდგომში.

თუ ცვლადი გამოცხადებულია როგორც მასივი (Count>1, მაგალითად, “LCD_Page”), მაშინ მისი სახელი მიუთითებს მასივის პირველ ბაიტზე. მასივის ელემენტის მიმართვისთვის უნდა გამოიყენოთ მისამართის წანაცვლება. მაგალითად, თუ საჭიროა ჩაწეროთ r0 მასივის მეხუთე ელემენტში, უნდა დაწეროთ:

```
r0->[@LCD_Page+5]
```

ქვემოთმოყვანილი მაგალითი წმინდავს LCD_Page მასივის ყველა ელემენტს:

```
Clear_LCD_Page
^ r0          r0 გაწმენდა
@LCD_Page -> Y  LCD_Page-ის მისამართის ჩაწერა Y-ში
  16 -> r20     ბაიტთა რაოდენობის მითითება
  ->|
  r0 -> [Y++]   მორიგე უჯრედში 0-ის ჩაწერა
    r20 --      ბაიტთა მთვლელი რიცხვის შემცირება
    -=|         თუ შედეგი არ უდრის 0-ს, გაგრძელება
  Ret
```

EEPROM ცვლადების გამოცხადების განყოფილება

განყოფილების დასახელება: EEPROM

ამ განყოფილებაში გათვალისწინებულია შემდეგი ველები:

- Name - გამოსაცხადებელი ცვლადის სახელი;
- Address - *(არააუცილებელი)* მუდმივა, რომელიც განსაზღვრავს მისამართის მნიშვნელობას. ნაგულისხმებად – ბოლოს განსაზღვრულის შემდეგი ან 0 განყოფილების დასაწყისში;
- Format - *(არააუცილებელი)* უჯრედის ფორმატი. ნაგულისხმებად მიიღება

ერთბაიტიანი ფორმატი. საჭიროების შემთხვევაში გამოიყენება მრავალბაიტიანი ფორმატის გამოიყენება;

- Count - *(არააუცილებელი)* გადანახული(რეზერვებული) უჯრედების რაოდენობა. ნაგულისხმებად 1-ს ტოლია;
- Value - *(არააუცილებელი)* საწყისი მნიშვნელობა, რომლებიც წარდგენილია მუდმივით ან მძიმით დაშორებული მუდმივთა მასივით(თუ Count >1) ;

  ასევე, შესაძლებელია საწყისი მნიშვნელობების გარე ფაილიდან ჩატვირთვა ბრძანებით "Load: FileName", სადაც FileName არის ამოსაკითხი ფაილის დასახელება

  იხ. "*მონაცემთა ფაილების უშუალო ამოკითხვა*".

თუ რაიმე ველი შეუვსებელი დარჩა, მის მნიშვნელობად აღებული იქნება $FF.

განყოფილების შევსების მაგალითი:

EEPROM:

Name	Address	Format	Count	Value	Comment
EE_Cell					ერთბაიტიანი უჯრედი
EE_LCD_Page			16	1,2,3,4,5,6,7,8	16 ერთბაიტიანი უჯრედი
InitValue		Word			ორბაიტიანი უჯრედი
StartValue		Word	3	259,3331,0	3 ორბაიტიანი უჯრედი საწყისი
					მნიშვნელობების ჩატვირთვით
InitScale		DWord		#h27773F	ოთხბაიტიანი უჯრედი საწყისი
					მნიშვნელობის ჩატვირთვით
FileTable		Word		Load: Table.db	მასივი ფაილიდან ჩატვირთვით

ალგორითმებში, გამოცხადებული სახელი შეიძლება გამოყენებული იყოს EEPROM ქმედების შემსრულებელ მაკრო-ოპერატორებში, მაგალითად:

```
InitValue -> X,        1875 -> InitValue
```

დასახელება პრეფიქსით '@' არის მუდმივა, რომელიც შეიცავს თავის მისამართს EEPROM-ში.

ქვემოთ მოყვანილია მასივის 'EE_LCD_Page' კოპირება SRAM მასივში 'LCD_Page' .

```
          ┌Read_From_EEPROM
@EE_LCD_Page.0 -> XL  EEPROM საწყისი მისამართის ჩაწერა X-ში
@EE_LCD_Page.1 -> XH
  @LCD_Page.0 -> YL   SRAM საწყისი მისამართის ჩაწერა Y-ში
  @LCD_Page.1 -> YH
      10 -> r20       ბაიტთა რაოდენობის მითითება (10)
      ┌──>┐
      │  XL -> EEARL      მიმდინარე EEPROM მისამართის ჩაწერა
      │  XH -> EEARH      X-დან ორმაგ რეგისტრში EEAR
      │   1 -> EERE       1-ს ჩაწერა ბიტში EERE
      │ EEDR -> r16       EEDR ჩაწერა r16-ში
      │ r16 -> [Y++]      r16-ის კოპირება SRAM-ის Y მისამართში
      │    X + 1          მიმდინარე EEPROM მისამართის 1-თ გაზრდა (X-ში)
      │   r20 --          ბაიტთა მთვლელი რიცხვის 1-თ შემცირება
      └──<-=>             თუ შედეგი არ არის 0, დაბრუნება ციკლის
         Ret
```

ქვემოთ მოყვანილია SRAM მასივიდან 'LCD_Page' კოპირება EEPROM მასივში 'EE_LCD_Page'.

```
          ¦Write_To_EEPROM
@EE_LCD_Page.0 -> XL  EEPROM საწყისი მისამართის ჩაწერა X-ში
@EE_LCD_Page.1 -> XH
  @LCD_Page.0 -> YL   SRAM საწყისი მისამართის ჩაწერა Y-ში
  @LCD_Page.1 -> YH
      10 -> r20       ბაიტთა რაოდენობის მითითება (10)
      ¦
      ¦<-- EEWE = 1   წინა ჩაწერის დასრულების მოლოდინი
      ¦   XL -> EEARL  მიმდინარე EEPROM მისამართის ჩაწერა
      ¦   XH -> EEARH  X-დან ორმაგ რეგისტრში EEAR
      ¦  [Y++] -> r16  SRAM-ში Y მისამართზე ბაიტის ჩაწერა r16-ში
      ¦   r16 -> EEDR  r16-ის ჩაწერა რეგისტრში EEDR
      ¦    1 -> EEMWE  ჩაწერის პროცესის
      ¦    1 -> EEWE   ინიციალიზება (დაწყება)
      ¦      X + 1     მიმდინარე EEPROM მისამართის 1-ით გაზრდა (X-ში)
      ¦     r20 --     ბაიტთა მთვლელი რიცხვის 1-ით შემცირება
      ¦--<  -=         თუ შედეგი არ არის 0, დაბრუნება ციკლის
          Ret
```

მაკრო-ოპერატორების გამოყენებით ეს მაგალითები უფრო ლაკონური იქნება:

```
          ¦Read_From_EEPROM
@EE_LCD_Page -> X
  @LCD_Page -> Y
     16 -> r20
      ¦
      ¦<-- EE[X++] -> [Y++]
      ¦--<  r20--
          Ret
```

```
          ¦Write_To_EEPROM
@EE_LCD_Page -> X
  @LCD_Page -> Y
     16 -> r20
      ¦
      ¦<-- [Y++] -> EE[X++]
      ¦--<  r20--
          Ret
```

მუდმივების წარდგენა ალგებრული ჩანაწერების სახით

მუდმივა, რომელელსაც ოპერატორთა შაბლონში აქვს სახე “#”, შეიძლება წარმოდგენილი იყოს არა მხოლოდ უშუალოდ, არამედ ალგებრული სახითაც კი.

ასეთი გამოსახულების წევრები შესაძლოა იყოს:

- უშუალოდ წარდგენილი მუდმივები;
- მუდმივები გამოცხადებული განყოფილებაში `Constants;`
- წვეროებისა და ჭდეების დასახელებები, რომლებიც შეიცავენ მეხსიერებაში სათანადო ადგილის მისამართს;
- დასახელებები გამოცხადებული განყოფილებებში SRAM და EEPROM სათანადო პრეფიქსით “@”, რომლებიც შეიცავენ მეხსიერებაში სათანადო ადგილის მისამართს;
- დასახელებები გამოცხადებული განყოფილებაში Bits რეგისტრთა ბიტები პრეფქსით “@”, რომლებიც შეიცავენ ბიტების ნომრებს;

გამოსახულებაში შესაძლოა გამოყეენბული იყოს არითმეტიკული მოქმედებები: აჯამვის “+”, გამოკლების “-”, გამრავლების “*” და მთელრიცხოვანი გაყოფისა “/”. გარდა ამისა, შესაძლოა ბიტური ოპერაციების შესრულება, როგორიცაა ***ან*** “!”, ***და*** “&” და გამომრიცხავი ***ან*** “^”. მაგალითად ოპერატორებში:

“`[@LCD_Page+ 5*3] -> r0`” (შაბლონი: “`[#]->R`”)

“`@LCD_Page+ 5*3`” – ალგებრული სახით წარდგენილი მუდმივაა.

შესაძლოა რთული გამოსახულებების აგება მრგვალი ფრჩხილების გამოყენებით.

ასევე, გათვალისწინებულია ხარისხში აყვანის ფუნქცია. სინტაქსისი: `A(B)`, სადაც `A` –

ხარისხის ფუძე, ხოლო B – ხარისხის მაჩვენებელი. მაგალითად, მიკროკონტროლერში AT90S8515 რეგისტრში TIFR ბიტს OCF1A აქვს ნომერი 6, ხოლო ბიტს OCF1B – ნომერი 5. ამ შემთხვევაში ოპერატორები

```
2(@OCF1A)+2(@OCF1B) ->r16
r16 ->TIFR
```

იქნება შემდეგის ტოლძალოვანი:

```
2(6)+2(5) ->r16 ან 64+32 ->r16 ან #b01100000 -> r16
r16 ->TIFR
```

ამით იწმინდება TIFR რეგისტრის ბიტები OCF1A და OCF1B.

გარდა ამისა, მისაწვდომია სტანდარტული გამოსახულებები, რომლებით წარდგენილი ცვლადთა ცხრილში გამოცხადებულ მუდმივები:

FormatOf(Var) – ცვლადის “Var” ბაიტთა რაოდენობა (SRAM, EEPROM, I/O ან სამუშაო რეგისტრი);

CountOf(Var) – გადანახული უჯრედების რაოდენობა ცვლადისთვის “Var” (ველი“Count”, SRAM ან EEPROM).

მაგალIთად, ზემოთ მოყვანილ მაგალითში გამოცხადებულია ცვლადი “Reg” format: Int24, Count: 4. ამ ცვლადისთვის გამოსახულება “FormatOf(Reg)” იქნება მუდმივა 3-ის ტოლი, ხოლო “CountOf(Reg)” რეგისტრი – მუდმივა 4-ის ტოლი...

მაკრო-ოპერატორები

დაპროგრამება მიკროკონტროლერის მხოლოდ და მხოლოდ ელემენტარული ბრძანებებით ხშირად გამოუსადეგარია. თუ, მაგალითად, საჭიროა ჩაწეროთ მუდმივა $1234 ორმაგ რეგიტრში X, მაშინ საჭიროა დაყოთ ის ე.წ უფროს და უმცროს ნაწილებად და რიგ-რიგობით ჩაწერა:

```
$12 -> XH
$34 -> XL
```

ამასთან იკარგება სიმარტივე. ცხადია, საჭიროა არსებობდეს შესაძლებლობა ეს მოქმედება რომ წარდგეს გასაგები სახით “$1234 -> X”, ხოლო ამწყობს უკვე უნდა შეეძლოს ამ აღნიშნვის დამუშავება და დაშლა ზემოთმოცემულ ნაწილებად. და თუ მუდმივა ათწილადის სახეობისაა, მაშინ აღნიშნვის შემოტანა გაცილებით უფრო მნიშვნელოვანია.

გარდა ამისა, ბევრი საჭირო ქმედების შესრულება შეუძლებელია მხოლოდ ელემენტარუ-ლი ოპერატორების გამოყენებით. მაგალითად, “37 -> r0” ან “$FF -> DDRB”. ორივე შემთხვევაში ჯერ მუდმივა უნდა იყოს ჩატვირთული რაღაც უფროს რეგისტრში, ხოლო მხოლოდ ამის მერე ჩაწერილი უნდა იყოს r0 ან DDRB-ში.

ამისთვის შემოღებული იქნა მაკრო-ოპერატორები. Algorithm Builder-ს აქვს შემდეგი მაკრო-ოპერატორების მხარდაჭერა:

არითმეტიკა-ლოგიკური მაკრო-ოპერატორები. გამოიყენება ელემენტში FIELD

შაბლონი	კომენტარი
*-> *	ასლის გაკეთება (კოპირება)
* + *	არითმეტიკული მიმატება
* - *	არითმეტიკული გამოკლება
* & *	ბიტური ქმედება **“და”**
* ! *	ბიტური ქმედება **“ან”**
* ^ *	ბიტური ქმედება “გამომრიცხავი **ან**”
^ *	ჩამოყრა (ნულის ჩაწერა)
* ++	ინკრემენტი (ერთით გაზრდა)
* --	დეკრემენტი (ერთით შემცირება)
- * -	ბიტური ინვერსია (შებრუნება)
- *	არითმეტიკული ინვერსია (შებრუნება)
* >>	ლოგიკური დაძვრა მარჯვნივ
> * >>	ლოგიკური დაძვრა მარჯვნივ გადატანით
± * >>	არითმეტიკული დაძვრა მარჯვნივ
<< *	ლოგიკური დაძვრა მარცხნივ
<< * <	ლოგიკური დაძვრა მარცხნივ გადატანით
* ->	ჩაწერა სტეკში
-> *	ჩაწერა სტეკიდან
# -> *.#	ბიტში ერთის ან ნოლის ჩაწერა
*.# -> *.#	წევრს შორის ბიტების ჩაწერა

მაკრო-პირობები. გამოიყენება ელემენტში CONDITION.

შაბლონი	კომენტარი
* = *	გადასვლა თუ ტოლია
*-= *	გადასვლა თუ ტოლი არ არის
* < *	გადასვლა თუ ნაკლებია
* > *	გადასვლა თუ მეტია
* <=*	გადასვლა თუ ნაკლები ან ტოლია
* >=*	გადასვლა თუ მეტი ან ტოლია
* ±<*	გადასვლა თუ ნაკლებია ნიშნის გათვალისწინებით
* ±>*	გადასვლა თუ მეტია ნიშნის გათვალისწინებით
±<=	გადასვლა თუ ნაკლები ან ტოლია ნიშნის გათვალისწინებით
±>=	გადასვლა თუ მეტი ან ტოლია ნიშნის გათვალისწინებით
* --	დეკრემენტი და გადასვლა თუ შედეგი ნულის ტოლი არ არის
*.# = #	გადასვლა თუ ოპერანდის ბიტი არის 0 ან 1

ვარსკლავით (*) აღნიშნულია თავისუფალი **წევრები** (ოპერანდები):

შაბლონი	კომენტარი
R	სამუშაო რეგისტრი (...)
P	შეტანა-გამოტანის რეგისტრი (...)
#	მუდმივა (...)
(SRAM)	გამოცადებული ცვლადი SRAM (...)
(EEPROM)	გამოცადებული ცვლადი EEPROM (...)
[#]	პირდაპირ მიმართული SRAM უჯრედი
[X]	ირიბად მიმართული SRAM უჯრედი, X-ით
[--X]	ირიბად მიმართული SRAM უჯრედი, X-ის პრე-დეკრემენტით
[X++]	ირიბად მიმართული SRAM უჯრედი, X-ის პოსტ-ინკრემენტით
[Y]	ირიბად მიმართული SRAM უჯრედი, Y-ით
[--Y]	ირიბად მიმართული SRAM უჯრედი, Y-ის პრე-დეკრემენტით
[Y++]	ირიბად მიმართული SRAM უჯრედი, Y-ის პოსტ-ინკრემენტით
[Y+#]	ირიბად მიმართული SRAM უჯრედი, Y-იდან # ბაიტით დაძვრით
[Z]	ირიბად მიმართული SRAM უჯრედი, Z-ით
[--Z]	ირიბად მიმართული SRAM უჯრედი, Z-ის პრე-დეკრემენტით
[Z++]	ირიბად მიმართული SRAM უჯრედი, Z-ის პოსტ-ინკრემენტით
[Z+#]	ირიბად მიმართული SRAM უჯრედი, Z-იდან # ბაიტით დაძვრით
EE[#]	პირდაპირ მიმართული EEPROM უჯრედი
EE[X]	ირიბად მიმართული EEPROM უჯრედი, X-ით
EE[X++]	ირიბად მიმართული EEPROM უჯრედი, X-ის პოსტ-ინკრემენტით
EE[Y]	ირიბად მიმართული EEPROM უჯრედი, Y-ით
EE[Y++]	ირიბად მიმართული EEPROM უჯრედი, Y-ის პოსტ-ინკრემენტით
EE[Z]	ირიბად მიმართული EEPROM უჯრედი, Z-ით
EE[Z++]	ირიბად მიმართული EEPROM უჯრედი, Z-ის პოსტ-ინკრემენტით
LPM[Z]	ირიბად მიმართული FLASH უჯრედი, Z-ით
LPM[Z++]	ირიბად მიმართული FLASH უჯრედი, Z-ის პოსტ-ინკრემენტით
ELPM[Z]	ირიბად მიმართული FLASH უჯრედი, Z-ითა და RAMPZ-ით
ELPM[Z++]	ირიბად მიმართული FLASH უჯრედი, Z-ის პოსტ-ინკრემენტითა და RAMPZ-ით

ოპერანდები, მონიშნული როგორც (...) შესაძლოა იყოს მრავალბაიტიანი. დასაშვებია სხვადასხვა ფორმატის ოპერატორების ჩასმა. ამასთან, თუ ოპერაციის შედეგის მიმღები ოპერანდი მეორეზე მოკლეა, მაშინ შედეგს ექნება უმოკლესის სიგრძე, ხოლო საწინააღმდეგო შემთხვევაში, ზედმეტი ბაიტები ნულებით იქნება შევსებული. მაგალითად, მაკრო-ქმედებები:

```
r16 -> Z                Z -> r16
```

გადაიწერება შემდეგი სახით:

```
r16 -> ZL               ZL -> r16
0 -> ZH
```

ოპერაციათა კორექტულობისთვის, ნიშანცვალებად სიდიდეებზე მოქმედებისას, ორივე ოპერანდი უნდა იყოს ერთსა და იგივე ფორმატის. წინააღმდეგ შემთხვევაში შედეგად მიღებულ

რიცხვს შესაძლოა არასწორი მნიშვნელობა ჰქონდეს. ხოლო მაკრო-პირობებში ოპერანდები უნდა იყოს მხოლოდ ერთსა და იგივე ფორმატის.

ირიბად მიმართული ოპერანდები არის ერთბაიტიანი. ირიბად მიმართულ მრავალბაიტიან ოპერანდებზე მოქმედებისთვის მითითებული უნდა იყოს მათი ფორმატები, მაგ.:

```
$AB3E -> [Y]:Word
```

ასეთი მაკრო-ოპერატორი გადაიწერება ოპერატორების შემდეგი მიმდევრობით:

```
$3E -> r16
r16 -> [Y]
$AB -> r16
r16 -> [Y+1]
```

მრავალბაიტიან ოპერატორებში მოქმედებები, მარჯვენა ბიტური დაძვრის გარდა, სრულდება უმცროსი ბაიტიდან დაწყებული (მარჯვნივ დაძვრისას – უფროსიდან). ამის გათვალისწინებითმ არსებობს გარკვეული შეზღუდვები ირიბი მიმართვის მქონე მაკრო-ოპერაციების შექმნისას:

- ოპერაციებში, რომლებიც იწყება უმცროსი ბაიტით შეუძლებელია ოპერანდებში პრე-დეკრემენტის გამოყენება, მაგ.: `[--X]:Word + 24000`
- ოპერაციებში, რომლებიც იწყება უფროსი ბაიტით (დაძვრა მარჯვნივ) შეუძლებელია ოპერანდებში პოსტ-დეკრემენტის გამოყენება: `[Z++]:Int24>>`

გარდა ამისა, შეუძლებელია მრავალბაიტიანი ოპერატორის შექმნა ოპერანდით `[X]`, რადგან რეგსტრისთვის `X` AVR ბრძანებათა სისტემაში არ არის განსაზღვრული წანაცვლებული მიმართვა (`[X+#]`). გამოყენებული უნდა იყოს `[X++]` ან `[X--]`

თუ მაკრო-ქმედების შექმნა აღმოჩნდება შეუძლებელი, მაშინ კომპილატორს გამოიტანს შემდეგ შეტყობინებას: “Such macro-operation can not be created ” (“ასეთი მაკრო-ოპერაციის შექმნა შეუძლებელია”)

EEPROM-ის ოპერანდების გამოყენებისას, კომპილატორი ავტომატურად ჩატვირთავს საჭირო კოდს, რომელიც უზრუნველყოფს მასში ჩაწერა/წაკითხვას. ამასთან გათვალისწინებული უნდა იყოს, რომ თუ ხდება EEPROM-ში ჩაწერა, მაშინ კოდს დართული ინქება ჩაწერის დასრულების დალოდება. მუშაობის დროს ეს მოითხოვს რამოდენიმე მილიწამს. EEPROM-ში ჩაწერის მაკრო-ოპერატორის სახე შეიძლება განსაზღვრული იყოს გამართვის სათანადო მენიუში (მენიუ “Options\Project Options” → “EE Macro”).

დაკომპილების დროს მაკრო-ოპერატორები გადაიწერება მიკროკონტროლერის ელემენტარულ ინსტრუქციათა კრებულის სახით. ამასთან, როგორც წესია, გამოიყენება რეგისტრები-”შუამავალები” `r16` და `r17`. ამიტომაც, პრობლემების თავიდან ასაცილებლად, მათი გამოყენება მაკრო-ოპერატორებთან ერთად *არასაურველია* და წყვეტის მომსახურე პროგრამებში ისინი *შენახული უნდა* იყოს დასტაში (stack).

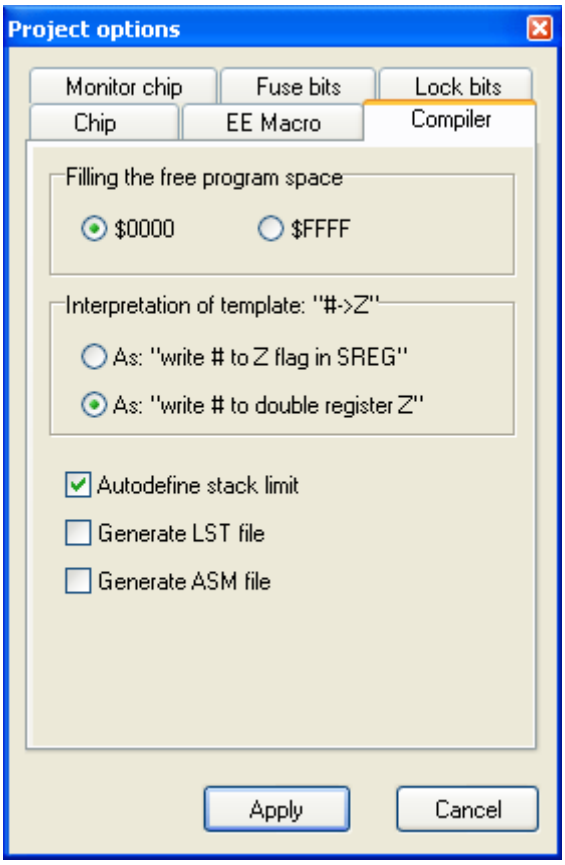
თუ მაკრო-ოპერატორში მოსალოდნელია მოქმედება მუდმივაზე, ამ შემთხვევაში აჯობებს სამუშაო რეგისტრების გამოყენება, რომლებიც გამოიცხადება უფროს ნახევარში. წინააღმდეგ შემთხვევაში გამოიყენება შუა-მოქმედებები, რომლებიც ჩაწერენ მუდმივას ნაწილებს რეგისტრში `r16`. ამან შეიძლება გამოიწვიოს კოდის ზომის გაზრდა, გაორმაგებაც კი. თუმცა ასეთი ჩაწერა შეცდომას არ წარმოადგენს.

ასევე, გათვალისწინებული უნდა იყოს ის, რომ მაკრო-ქმედებების მიერ გამოყენებული (აღძრული) *დროშები* (flags) არ ემთხვევა ანალოგიურ დროშებს ელემენტალური ქმედებების დონეზე. მათი გამოყენება პირობითი გადასვლის ელემენტებში ან გადატანისთვის უმართებულოა.

ციკლთა განხორციელებისთვის მოსახერხებელია მაკრო-პირობის გამოყენება: “`*--`”. ის ერთდროულად შეიცავს ოპერანდის დეკრემენტს და განშტოებას, თუ შედეგი ნულის ტოლი არ არის.

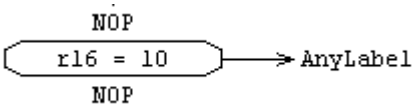
შენიშვნა: შაბლონს “#->Z” შეიძლება ჰქონდეს ორი გაგება:

“#-ს ჩაწერა SREG-ის Z ბიტში” ან “#-ს ჩაწერა ორმაგ რეგისტრში Z”. ამ გაუგებრობსგან თავის დასაღწევად უნდა ავირჩიოთ თუ რას ვგულისხმობთ პროექტის პარამეტრების ფანჯარაში:

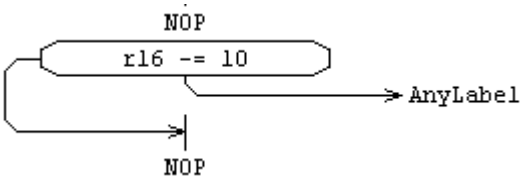


თუმცა, თუ ოპერატორის შრიფტს გავამსხვილებთ ღილაკით **F2**, მაშინ ის წაიკითხება როგორც: “#-ს ჩაწერა ორმაგ რეგისტრში Z”, ხოლო შაბლონი “#->.Z” კი წაიკითხება მხოლოდ როგორც: “#-ს ჩაწერა SREG-ის Z ბიტში”. ეს მოხდება დაყენებული პარამეტრების მიუხედავად.

წესით, პირობითი გადასვლის სიშორე AVR ელემენტალურ ოპერატორს შორის არ უნდა აღემატებოდეს ±63 ქმედებას. თუ ნამდვილი დაშორება აღემატება ამ რიცხვს, ამწყობი ავტომატურად გამოიყენებს უფრო რთულ, ინვერსული პირბითა და უთუო გადასვლისაგან შემდეგარ კონსტრუქციას. მაგალითად პირობითი გადასვლა:



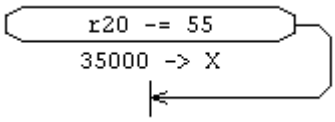
აიწყობა შემდეგნაირად:



მაკრო-ოპერატორის გამოყენებას არ აქვს დიდი გავლენა პროგრამის სრაფქმედებაზე, რადგან მათში ისედაც სრულდება ყოველივე ის, რისი შესრულება საჭიროა ელემენტარული ქმედებების გამოყენებისას.

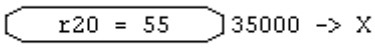
პირობითი ოპერატორები

ხშირად, რაღაც პირობის შესრულებისას საჭიროა მხოლოდ ერთი ქმედების შესრულება. მაგალითად:



აქ თუ `r20=55`, მაშინ `35 000` ჩაიწერება `X`-ში.

ასეთ შემთხვევებში უფრო მოსახერხებელია პირობითი ოპერატორის გამოყენება. ამ შემთხვევაში პირობის ადგილას იწერება შებრუნებული პირობა, ვექტორი(ისარი) არ უნდა იყოს გამოჩენილი, ხოლო მის ადგილას უნდა ჩაწერილი იყოს ოპერატორი:

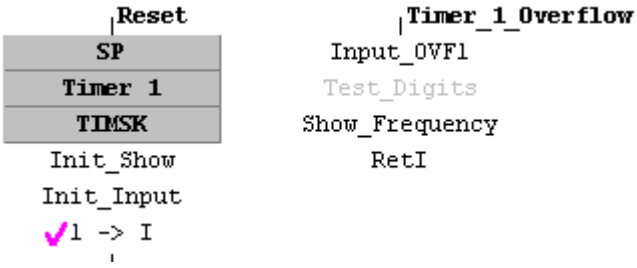


წყვეტის მომსახურე ბლოკები

დაპროგრამების პროცესის მოხერხებულობისთვის Algorithm Builder-ს აქვს ჭდეების განსაკუთრებული სახეობის მხარდაჭერა – წყვეტის მომსახურე ჭდეები. წყვეტის მოსახდენად, ჩვეულებრივ, საჭიროა უთუო გადასვლის ვექტორში სათანადო ქვეპროგრამის მისამართის მითითება. განსაკუთრებული ჭდეების მეშვეობით ამწყობი ყოველივე ამას აქეთებს ავტომატურად. ამისთვის საჭიროა ბლოკის წვეროს მიანიჭოთ წყვეტის სტანდარტული სახელი და მონიშნოთ ის, როგორც მაკრო ღილაკით **F2**. ამის შედეგად წვეროს შრიფტი უნდა გამსხვილდეს. იგივე მიიღწევა ფანჯრის მენიუდან: “Elements\Interrupt vectors\...”

თუ ამწყობს შეხვდება თუნდაც ერთი ასეთი ბლოკი, ის შეავსებს წყვეტის ვექტორების სივრცეს ქვეპროგრამიდან დაბრუნების კოდით (“RETI”), ხოლო წყვეტის სათანადო მისამართზე ჩაწერს უთუო გადასვლის ვექტორს, რომელიც მიმართულია მომსახურე ბლოკზე.

თუ მომსახურე ბლოკებს იყენებთ, მაშინ პროგრამის საწყისი მისამართები ავტომატურად დაკავებული იქნება მომსახურე ქვეპროგრამებზე უთუო გადასვლებით. ამიტომ, პროგრამის მართობულად დაწყების მიზნით, ალგორითმი უნდა დაიწყოს მაკრო-ჭდე “Reset”. ეს უზრუნველყოფს ნულოვან მისამართში უთუო გადასვლის ჩასმა, რომლის ვექტორი მიუთითებს პროგრამის დასაწყისზე. მაგალითად:



თუ ალგორითმი ისეთნაირად არის აგებული, რომ შესაძლოა რაიმე პროცესის შემთხვევით გაწყვეტა, წყვეტის მომსახურე პროგრამაში საჭიროა მოხდეს რეგისტრების, ბიტების და სხვა ფაქიზი მონაცემების შენახვა, რადგან წყვეტის შედეგად შეიძლება მოხდეს მონაცემის დაზიანება.

მაშასადამე, წყვეტის შექმნის დროს, აუცილებელია შემდეგის გაკეთება:

1. წვეროს შექმნა სახელით "`Reset`", რომლითაც დაიწყება პროგრამის შესრულება.
2. სტეკზე მიმთითებლის დასმა მიმნიჭებლით "`SP`" (ჩვეულებრივ, ეს არის SRAM-ის მაქსიმალური მისამართი)
3. სასურველი წყვეტის დაშვება (წამზომებისთვის – TIMSK რეგისტრის სათანადო ბიტები)
4. წყვეტის გლობალურად დაშვება ოპერატორით "`1 -> I`"
5. შეიყვანოთ წყვეტის მომსახურე პროგრამა, რომლის წვეროს ერქმევა წყვეტის დასახელება, ხოლო ბლოკის ბოლოში *აუცილებლად* მოთავსებული იქნება ოპერატორი

"RetI"

ჩამტვირთავი (boot) სექციის წყვეტისთვის, გამოიყენეთ წყვეტები, რომლების სახელები იწყება თავსართით "BOOT_" .

მონაცემთა პირდაპირი განთავსება პროგრამის მეხსიერებაში

პროგრამაში შესაძლებელია როგორც კოდის, ასევე რაღაც მონაცემთა ჩაწერა: კოდების ცხრილები, შეტყობინებათა სტრიქონები და ა.შ. ეს მონაცემები უნდა იყოს მოთავსებული საბოლოო ოპერატორების "RET", "RETI" ან უთუო გადასვლების შემდეგ. მათი ჩაწერისთვის გამოიყენება ელემენტი "FIELD". ამ ელემენტის მონაცემთა რეჟიმში გადაყვანა ხდება ღილაკთა კომბინაციით "SHIFT+F3" ან მენიუდან "Elements\Data", ამასთან ელემენტის მარცხენა მხარეს გაჩნდება მცირე შავი ოთხკუთხედი. მონაცემებზე *მისამართის შენახვისთვის* საჭიროა მათ წინ წეროს ან ჭდეს დასმა.

მონაცემების ჩაწერის წესი:

[ფორმატი] #, #, # ...

მონაცემთა ფორმატის მითეთება აუცილებელი არ არის, თუმცა ამ შემთხევაში ჩაითვლება, რომ მონაცემი ერთბაიტიანია. მაგალითად:

```
          |DataA
▪ 12, 14, 37, 55
          |DataB
▪ Word $ADCE, $5540, $3277, $5511
▪ Word $2E45, $D899, $E537, $4799

       |Read_DataA  მონაცემების კოპირების ქვეპროგრამა
DataA*2 -> Z       პროგრამის მეხსიერების საწყისი მისამართი
@XArray -> Y       SRAM  საწყისი მისამართის ჩატვირთვა
  4 -> r20         ბაიტთა რაოდენობის მითითება
          |
LPM[Z++] -> [Y++]  პროგრამის მეხსიერებიდან ბაიტის SRAM-ში ჩაწერა
    r20--          r20-ის შემცირება, გადასვლა თუ შედეგი არ არის 0
    Ret            ქვეპროგრამიდან გასვლა
```

მონაცემების ამოკითხვა ქმედებით "LPM" რეგისტრში Z, აუცილებელია გადაცემული იქნას **ორმაგი** მისამართი, რადგან ეს მოქმედება ითვალისწინებს პროგრამის მეხსიერების ბაიტურ დალაგებას.

გასათვალისწინებელია, რომ ამწყობი ამრგვალებს ბაიტთა ოდენობას ლუწამდე. თუ ბაიტთა ოდენობა კენტია, ბოლო ბაიტი შეივსება ნულით.

შესაძლებელია მონაცემების მოთავსება სტრიქონის სახით. ამასთან, ფორმატი უნდა იყოს მხოლოდ ერთბაიტიანი. სტრიქონი უნდა იყოს ჩაწერილი ორ ორმაგ ბრჭყალს შორის. მაგალითად:

"Hello!"

ამ შემთხვევაში მეხსიერება შეინახება სათანადო სიმბოლოების ANSI კოდები.

ზოგიერთ შემთხვევაში, აწყობილ მოწმყობილობას შეიძლება ქონდეს შეხება არა-ANSI კოდებთან. ამ შემთხვევაში სიმბოლოთა კოდები იცვლება ე.წ. მაშიფრებელი ფაილის მეშვეობით, რომლის სახელი მითითებულია მეორე ორმაგი ბრჭყალის მერე, მრგვალ ფრჩხილებში. Algorithm Builder-ს მოყვება ფაილი LCD_CYR.dcd, რომელსაც გადაჰყავს სიმბოლოთა კოდები თხევად-კრისტალიანი დისპლეის რუსული ანბანის კოდებში. მაგალითად:

"ПРОЦЕСС" (LCD_CYR)

დაშვებულია მონაცემთა შერეული ჩაწერა. მაგალითად:

```
"HELLO!", $0,  $DB
```

მონაცემთა ფაილის პროგრამის მეხსიერებაში ჩართვა

Algorithm Builder იძლევა საშუალებას ფაილის ჩართვა როგორც პროგრამის ნაწილი, ასევე როგორც საწყისი მნიშვნელობები EEPROM-ისთვის.

ამისთვის გამოიყენება ჩანაწერი: "Load: FileName", სადაც FileName – სასურველი ჩასართავი ფაილის სახელია. ამასთან, ფაილი შეიძლება იყოს შემდეგი ფორმატის:

- `IntelHEX` (გაფართოება ".hex");
- `General` (გაფართოება ".rom");
- `Binary` (გაფართოება ".bin");
- `Algorithm Builder-ის მონაცემების` ფაილთა ფორმატი გამოიყენება ნებისმიერი გაფართოება, მაგ ".db". ზოგადად, ეს არის ტექსტური ფაილი, რომელშიც ჩაიწერება მონაცემები ზემოთხსენებული ფორმატის მიხედვით. შედეგად, ეს მონაცემები ჩაიტვირთება უშუალოდ პროგრამის მეხსიერებაში.

მაგალითად:

```
$11, $22, $33
$44, $55, $66, $77,  "HELLO!"
Word $ABCD, $FEDC
```

პროექტში ალგორითმის ჩართვა გარე ფაილებიდან

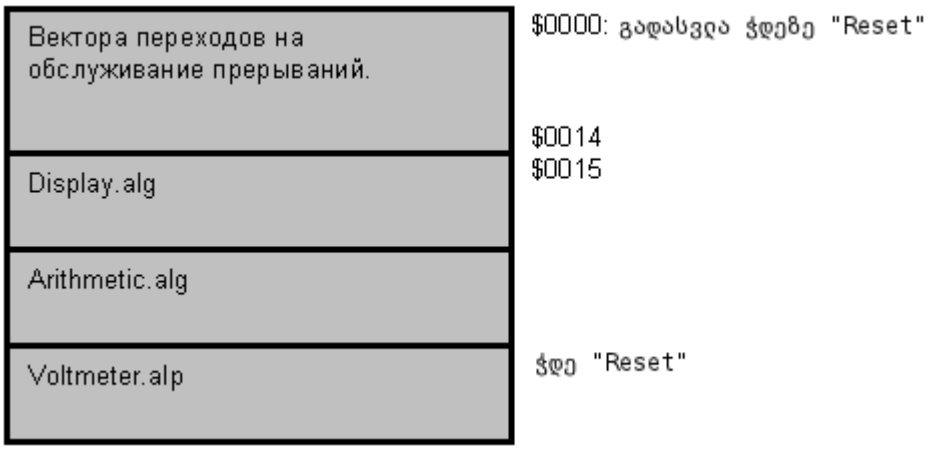
ახალი პროექტის შექმნის შედეგად იქნება მთავარი ფაილი გაფართოებით ".alp". ეს ფაილი ყოველთვის ნაჩვენებია რედაქტორის მარხენა ჩანართში. შესაძლოა პროექტი შემოიფარგლოს მხოლოდ ამ ფაილით, მაგრამ ბევრად უფრო მოსახერხებელია პროექტის რამდენიმე ფაილად დაყოფა, რომლებშიც ფუნქციები დაჯგუფებული იქნება მათი დანიშნულები მიხედვით. ამის გარდა, შესაძლოა ადრე შემუშავებული და გამართული ფუნქციების გამოყენება არსებულ პროექტში მათი ხელახალი დაწერის გარეშე.

ახალი ფაილის დამატებისთვის, არიჩიეთ მენიუს ელემენტი "File\New". ამასათან, პროგრამის ფანჯარაში გაჩნდება დამატებითი ჩანართი.

Algorithm Builder მოითხოვს, რომ პროექტის ყველა ფაილი მოთავსებული იყოს მთავარი .alp ფაილის საქაღალდეში. ამიტომ, პროექტში არსებული ალგორითმის დამატებისთვის, უპირველეს ყოვლისა, უნდა მოათავსოთ მისი ასლი სათანადო საქაღალდეში და მერე გახსნათ ის მენიუდან "File\Open", ან ხელსაწყოთა ზოლზე მწოფი ღილაკით  .

თუ რომელიმე ფაილი გახსნილია რედაქტორში, ეს სულაც არ ნიშნავს იმას, რომ ფაილი პროექტში ჩართული არის.

ფაილის პროექტში ჩართვა ხდება ამწყობის მითითების ჩაწერით: "`+: FileName`", სადაც `FileName` – ფაილის სახელი (ბრჭყალების გარეშე). მისი ჩაწერისთვის გამოიყენება ელემენტები "TEXT" ან "FIELD". ამწყობს რომ შეხვდება ასეთი მითითება, ის შეაჩერებს მიმდინარე ფაილის დამუშავებას და გადავა მითითებულ ფაილზე.

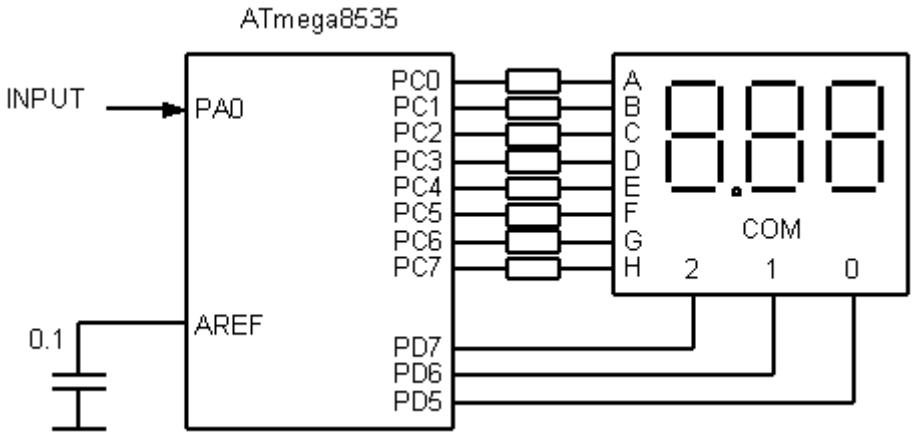


ქვემოთმოცემული *სიხშირეთმზომის* მაგალითი დაყოფილია სამ ფაილად: "Voltmeter.alp", "Display.alg" და "Arithmetic.alg". ამ პროექტის "+" მითითებების განლაგებით ხდება პროგრამის შემდეგნაირად შედგენა:

დირექტივების (მითითებების) განლაგების მართვით ხდება პროგრამის მეხსიერების დაწყობა.

მაგალითი - ვოლტმეტრი

ეს მაგალითი უფრო სრულად აღწერს Algorithm Builder-ის შესაძლებლობებს.



ვოლტმეტრი ზომავს ძაბვას 0-დან 2 ვოლტამდე, 10 მვ (mV) გარჩევადობით და აჩენებს მნიშვნელობას სამსეგმენტიან შუქდიოდურ დისპლეიზე საერთო კათოდით, დინამიური ჩვენების რეჟიმში.

ამ ვოლტმეტრის ალგორითმი მოთავსებულია საქაღალდეში "EXAMPLES\VOLTMETER".

პროექტი მოიცავს სამ ფაილს: "Voltmeter.alp" – პროექტის მთავარი ფაილი, "Display.alg" და "Arithmetic.alg".

ტაქტური სიხშირის გენერატორად დაყენებულია შიგა RC გენერატორი სიხშირით 1მგჰც (Fuse bits CKSEL = 0001).

ალგორითმის შესრულებაიწყება ჭდით "Reset" გვერდზე "Voltmeter". შემდეგ მიმნიჭებელი "SP" *დასტაში* (stack) მიმთითებელში ჩაწერს SRAM-ის საბოლოო მისამართს $25F, ქვეპროგრამების გაშვების უზრუნველყოფის მიზნით. მიმნიჭებელი "Timer 0" განსაზღვრავს *მთვლელი 0*-ის სიხშირეს როგორც CK/8, რაც იწვევს გადავსების პერიოდს 2.048 მწმ. მიმნიჭებელი "TIMSK" ნებას რთავს წყვეტას *მთვლელი 0*-ის გადავსების შემთხვევაში.

მერე გამოიძახება ქვეპროგრამა "Init_Display", რომელიც მოთავსებულია გვერდზე "Display". ეს ქვეპროგრამა ხდის გამომავლად **C** პორტს სრულად და **D** პორტის 5, 6 ,7 ბიტს, ჩამოყრის გამოსახული სეგმენტის ნომერს(ცვლადი "DigitIndex") და ასუფთავებს ციფრების მნიშვნელობას (სამბაიტიანი მასივი "Digits").

ოპერატორი "1->I" ნებას რთავს შესრულდეს გლობალური წყვეტა. შემდეგ ალგორითმი მუშაობს მხოლოდ წყვეტების მეშვეობით.

მთვლელი 0-ის გადავსებისას, პერიოდით 2.048 მწმ, გამოიძახება ქვეპროგრამა "**Timer_0_Overflow**", რომელშიც გამოიძახება ქვეპროგრამა "ShowNextDigit" გვერდიდან "Display". ჭდეში "DigitCodes" მოთავსებულია შვიდსეგმენტიანი ციფთა კოდები 0-დან 9-დე. ქვეპროგრამა "ShowNextDigit", ყოველ 2.048 მწმ-ს, ციკლურად, ცვლის ჩართული ინდიკატორის ინდექსს (ცვლადი "DigitIndex") 0-დან 2-დე, მასივიდან "Digits" ამოიღებს ერთ-ერთ ციფრს, გადაყავს ამ ციფრის შესაბამის შვიდსეგმენტიან კოდში და აგზავნის განახლებულ მნიშვნელობას **C** პორტში. ამავდროულად, იცვლება დისპლეის აქტიური კათოდი, რომელსაც გადაჰყავს ლოგიკური 0 ერთ-ერთზე შემდეგი ბიტებიდან PD5-PD7. ამგვარად, მიახლოებით ყოველ ორ მილიწამს დისპლეიზე იცვლება ანთებული ციფრი.

ანათვლის აღების (გარდაქმნის) შესრულების შემდეგ, **აცგ** მოახდენს წყვეტას, რომლის მომსახურე პროგრამა არის "ADC_Complete". ამ ქვეპროგრამაში გარდაქმნის შედეგი მრავლდება კოეფიციენტზე "k_ADC" ქვეპროგრამით "fMul_XY" გვერდიდან "Arithmetic". ეს ქვეპროგრამა უზრუნველყოფს ორმაგი რეგისტრების X და Y გადამრავლებას, შედეგი კი ჩაიწერება Z-ში. გამვრავლება ხდება ფორმულით: X * Y / 65536 -> Z. კოეფიციენტზე გამრავლებით მიიღება გარდაქმნის შედეგი მილივოლტების ათეულებში. მამასშტაბირებელი კოეფიციენტი "k_ADC" მიიღება "Voltmeter" გვერდზე მოთავსებული ცხრილიდან.

გარე ვოლტმეტრით შემავალზე "AREF" წინასწარ უნდა განისაზღვროს საბაზო ძაბვა მილივოლტებში (მუდმივა "Vref").

შემდეგ, ქვეპროგრამა "Z_to_Digits" ამოიღებს მიღებული(გაზომილი) რიცხვიდან ასეულები, ათეულები და ერთეულები, და ჩაწერს მათ "Digits" მასივის სათანადო ნაწილებში. შემდეგ კი ხდება მათი დისპლეიზე გამოსახვა.

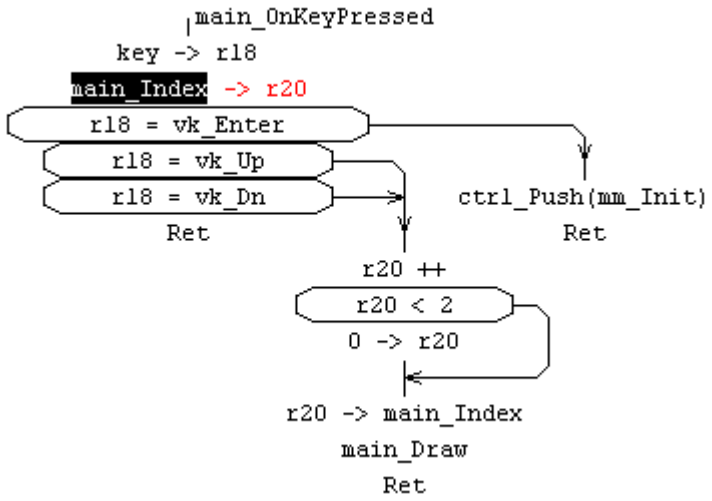
ალგორითმის დამუშავება

დასამუშავებელი ელემენტის არჩევა ხდება ღილაკელბით "↑" ან "↓" ან თაგვის მარხენა წკაპით.

ფრაგმენტის მონიშვნა.

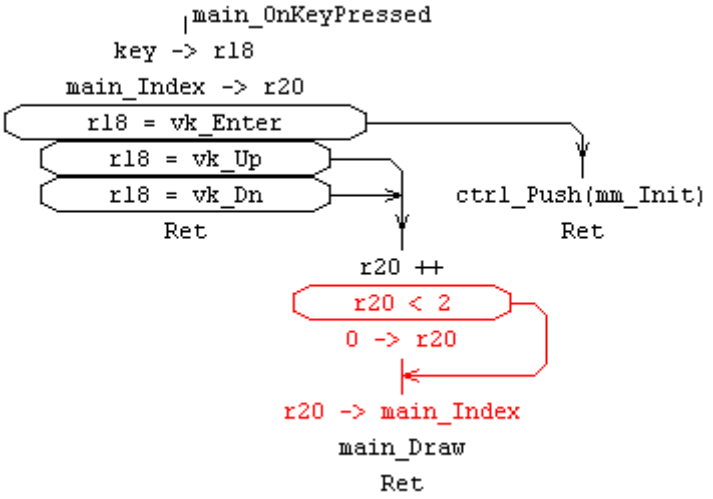
შესაძლებელია იყოს სამი სიტუაცია.

1. ფრაგმენტის მონიშვნა *სტრიქონის შიგნით*:



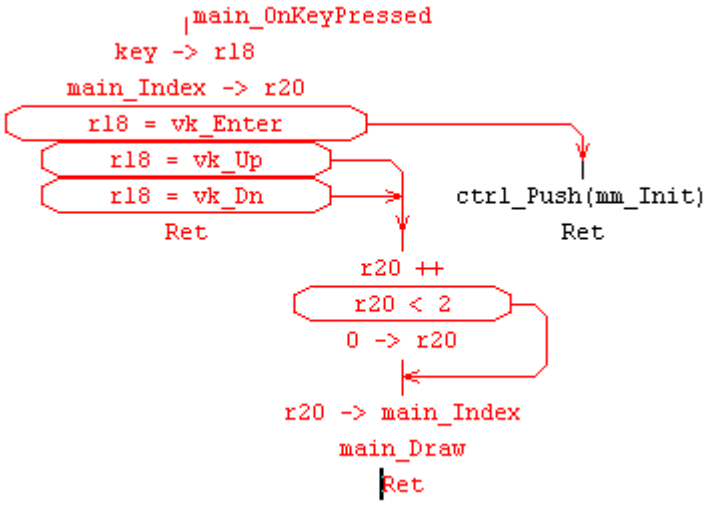
გამოიყენეთ ღილაკები "Shift+ ←" ან "Shift+ →"

2. ფრაგმენტის მონიშვნა *ბლოკის შიგნით*:



გამოიყენეთ ღილაკები "Shift+ ←" ან "Shift+ →".
ამასთან, მონიშნულ ელემენტებს შორის არ უნდა იყოს *ბლოკის წვერო*.

3. *ბლოკების მთლიანად* მონიშვნა:



გამოიყენეთ ღილაკები "Shift+ ↓" ან "Shift+ ↑".

ამ შემთხვევაში, მონიშნულ ელემენტებს შორის უნდა იყოს სულ მცირე ერთი წვერო. გარდა ამისა, ბლოკების მონიშვნა შესაძლებელია სარკმლის მეშვეობით, რომელიც ჩნდება თაგვის მარცხენა წკაპისა და "Shift" ღილაკის ერთდროული დაჭერის და გადაზიდვის დროს. ასევე, ცალკეული ბლოკის მონიშვნა შეიძლება ბლოკზე თაგვის მარხენა წკაპისა და "Ctrl" ღილაკის ერთდროული დაჭერით.

ნაგულისხმევად, აქტიური ელემენტი ითვლება მონიშნულ ბლოკის შიგა ფრაგმენტად.

მონიშნული ფრაგმენტების ან ელემენტების **ასლის** ბუფერში მოთავსებისთვის, გამოიყენეთ "Ctrl+C" ან "Ctrl+Insert".

ფრაგმენტების **წაშლისთვის** გამოიყენეთ "Ctrl+Delete", ხოლო **ამოჭრისთვის** - "Ctrl+X" ან "Shift+Delete".

ჩასმისთვის გამოიყენეთ ღილაკები "Ctrl+V" ან "Shift+Insert". ამ შემთხვევაში შესაძლოა იყოს 3 ვითარება.

1. თუ ბუფერში *სტრიქონის* ფრაგმენტია, მაშის ის ჩაისმევა სტრიქონში, სადაც იმყოფება კურსორი;
2. თუ ბუფერში *ბლოკის* ფრაგმენტია, მაშინ ის ჩაისმევა აქტიური სტრიქონის ქვეშ.
3. თუ ბუფერში *ერთი ან რამდენიმე ბლოკია*, მაშინ ფანჯარაში გამოისახება ჩასასმელი ბლოკის კონტური. თაგვის მოძრაობით ან მიმართულების ღილაკებით არიჩიეთ ჩასასმელი ბლოკის სასურველი მდებარეობა. ბლოკის დასმა ხდება თაგვის მარცხენა ღილაკით. ჩასმის გაუქმებისთვის გამოიყენება ღილაკი "Escape"

შეჩერების წერტილის, ე.წ. "Break Point", დასმის ან მოხსნისთვის, გამოიყენეთ "F5".

ელემენტის **მაკრო-ოპერატორად** და უკან გადაქცევისთვის, გამოიყენეთ ღილაკი "F2". მაკრო-ოპერატორები გამოირჩევიან **მსხვილი** შრიფტით.

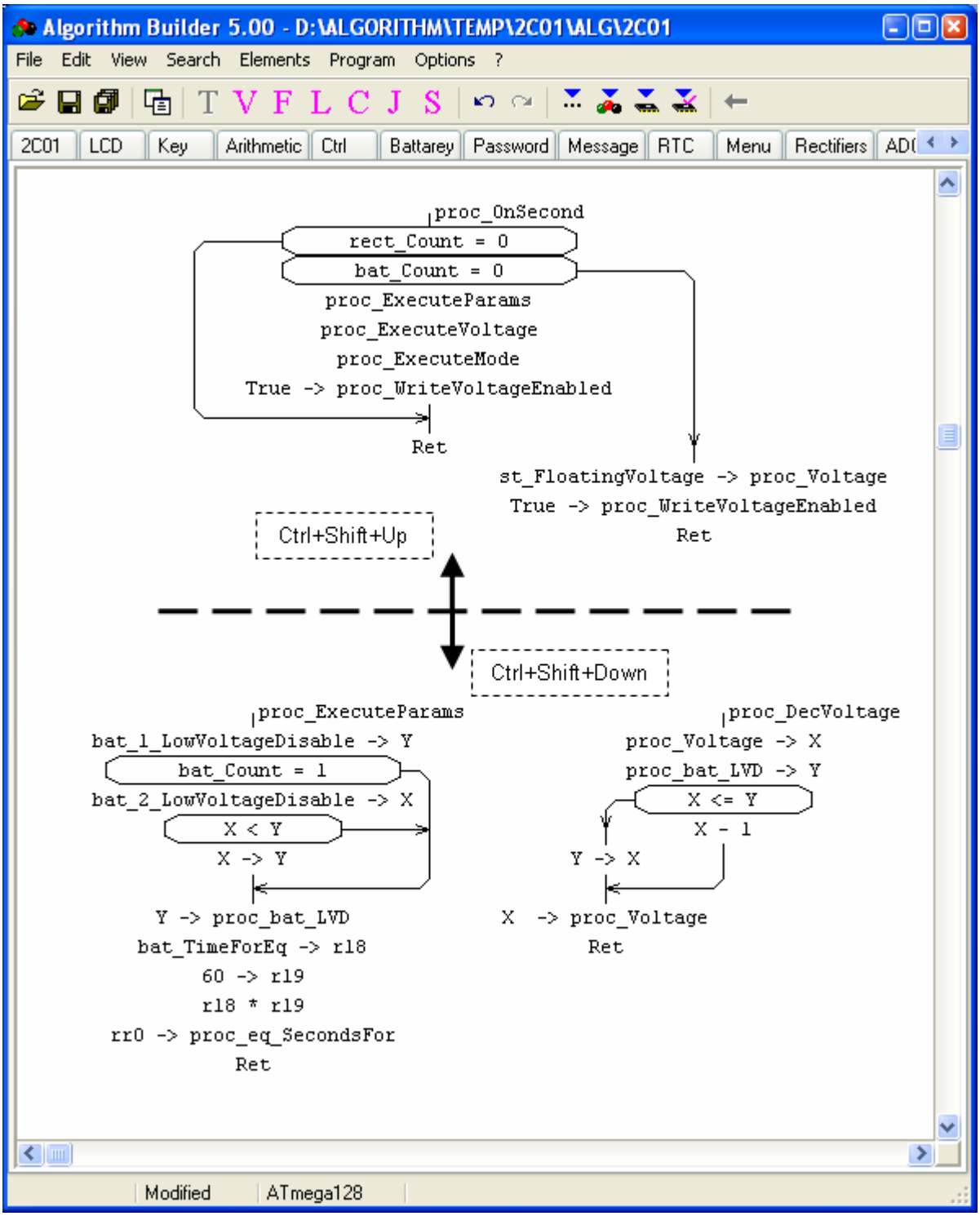
ალგორითმის სამუშაო ველის **გადახვევისთვის**, გამოიყენეთ გადახვევის ზოლები ან თაგვის ჩაჭირებული მარცხენა ღილაკი. გარდა ამისა, შესაძლოა თაგვის ბორბლის გამოყენება ვერტიკალური გადახვევის მიზნით.

კურსორის ბლოკთა წვეროებში მოთავსებისთვის გამოიყენეთ ღილაკები "Page Up" და "Page Down".

ალგორითმის **დასაწყისში** გადასვლისას, გამოიყენეთ "Ctrl + Page Up", ხოლო **ბოლოში** - "Ctrl + Page Down".

იმ ბლოკის გადატანა, რომელშიც მოთავსებულია კურსორი, ხდება "Ctrl" ღილაკთან ერთად მიმართულების ღილაკების დაჭერით, ან გადაზიდვით ბლოკზე მარხენა წკაპის მეშვეობით.

აქტიური და ყველა მომდევნო ბლოკის ზემოთ ან ქვემოთ გადატანისთვის გამოიყენება "↓" ან "↑" ღილაკები "Ctrl+Shift" კომბინაციასთან ერთად. ამის გამოყენება შესაძლებელია პროექტში ვერტიკალური ადგილის გასუფთავებისთვის:



მონიშნულ ბლოკთა ჯგუფების გადატანა ხდება თაგვის მარცხენა ღილაკით გადაზიდვით, ან ღილაკებით "↓ ↑ ← →". მონიშვნის გაუქმებასა და გადატანის დასრულებისთვის დააჭირეთ ღილაკს "Escape".

უკუქმედებისთვის, ანუ შესრულებული მოქმედებების *გაუქმებისთვის*, გამოიყენეთ კომბინაცია "Alt+უკუშდა" (იგივე "Alt+Backspace"), ან ღილაკი .

ბლოკების მეხსიერებაში გადალაგება სიჩქარის ამაღლების მიზნით შეიძლება

რამდენიმე ბლოკის მონიშვნით და დაჯგუფებით. ბლოკების ამორჩევა უნდა მოხდეს იმ თანმიმდევრობით, რა თანმიმდევრობითაც უნდა მოხდეს მათი შესრულება. ყველაზე მოხერხებულია ბლოკების მონიშვნა მარცხენა წკაპისა და Ctrl კლავიშთა კომბინაციით. ბლოკების მონიშვნა რომ დასრულდება, უნდა აირჩეს პუნქტი "Group selected box" მენიუდან "Edit".

ჯგუფში შესული ბლოკის დამუშავებისთვის ორჯერ დაწკაპეთ მასზე. უკან დაბრუნებისთვის გამოიყენეთ ღილაკი ან პუნქტი "Back" მენიუდან "Edit".

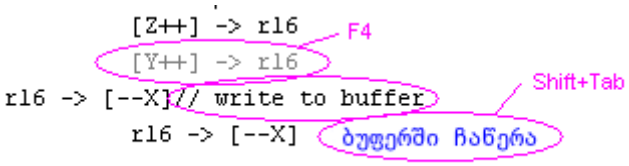
კომენტარები

შესაძლებელია რამდენიმე სახის კომენტარების გაკეთება.

• აქტიური ელემენტის გათიშვა შეიძლება ღილაკით "F4". ნაგულისხმევად, ასეთი ელემენტი გახდება ნაცრისფერი და ამწყობი მას უგულებელყოფს. "F4" ღილაკის მეორედ დაჭერით ელემენტი ისევ ჩაირთვება.

• ამწყობი ასევე უგულებელყოფს ორი დახრილი ხაზის შემდგომ ტექსტს "//".

• ელემენტისთვის მიკერძოებული კომენტარის გაკეთებისთვის, გამოიყენება ღილაკთა კომბინაცია "Shift+Tab".



რესურსთა განაწილების ცხრილის დამუშავება

ცხრილში ჩამსწორებლის ფოკუსის სხვადასხვა ელემენტებზე გადატანისთვის გამოიყენეთ ისრიანი ღილაკები ან "Tab" (მომდევნო უჯრედში გადასვლა) ან ღილაკები "Shift+Tab" (წინა უჯრედში გადასვლისთვის), ან თაგვის მარცხენა ღილაკით უჯრედზე დაწკაპით.

ახალი სტრიქონის **დამატებისთვის** (აქტიურის ქვემოთ), გამოიყენება ღილაკი "Enter", ხოლო ახალი სტრიქონის დამატებისთვის აქტიურის ზემოთ – "Insert".

რამდენიმე სტრიქონის **მონიშნვისთვის**, გამოიყენება ღილაკები "Shift+↑" და "Shift+↓". ნაგულისხმევად, აქტიური სტრიქონი მონიშნულად ითვლება.

აქტიური სტრიქონის **წაშლისთვის** გამოიყენეთ "Ctrl+Del", ხოლო სტრიქონის **ამოჭრისთვის** – "Shift+Del" ან "Ctrl+X".

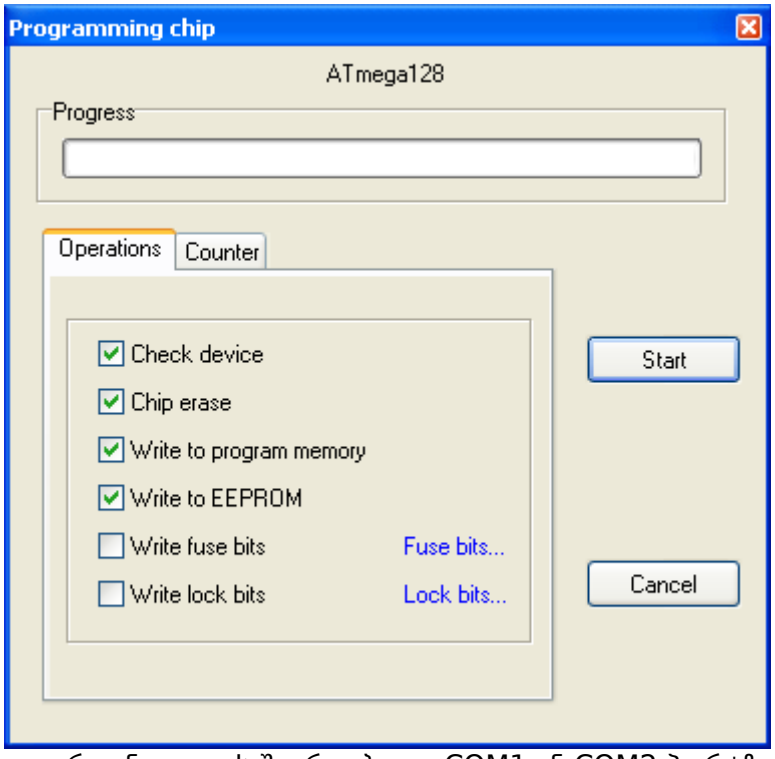
მონიშნული სტრიქონების ასლების ბუფერში ჩასმითვის გამოიყენება "Ctrl+C" ან "Ctrl+Insert".

ბუფერიში მყოფი სტრიქონების ჩასმისთვის გამოიყენება "Ctrl+V" ან "Shift+Insert". ამასთან, სტრიქონები ჩაისმება აქტიური სტრიქონის ქვემოთ. მუშაობისას გასათვალისწინებელია, რომ ბუფერიდან ჩასმა შესაძლებელია მხოლოდ *სექციის ფარგლებში*, ან სხვა გვერდის *იგივე სექციაში*.

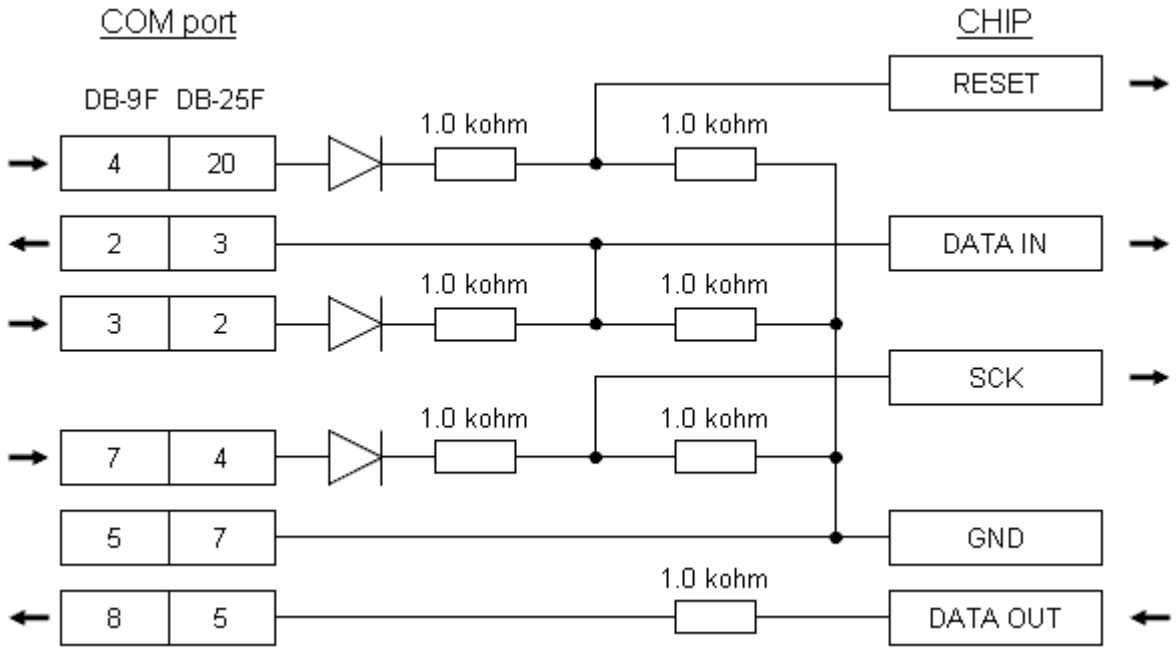
უჯრედის სიფართის შეცვლისთვის მიიტანეთ ისარი სვეტის სათაურის კიდესთან, დააჭირეთ თაგვის მარცხენა ღილაკს და აუშვებლად, გადაადგილეთ იგი.

მიკროკონტროლერის დაპროგრამება

Algorithm Builder შეიცავს ჩაშენებულ შიგასქემურ პროგრამატორს, რომელიც უზრუნველყოფს მიკროსქემების მიმდევრობით დაპროგრამებას. თუ აირჩევთ მენიუს პუნქტს "Program\Run With Chip" ან დააჭერთ ღილაკებს "Ctrl+Shift+F9" ან  ხელსაწყოთა პანელიდან, დაიწყება ალგორითმის აწყობა (დაკომპილება). თუ ეს წარმატებით დასრულდა, გაიხსნება დაპროგრამების ფანჯარა:



მიკროკონტროლერი უნდა იყოს შეერთებული COM1 ან COM2 პორტზე მარტივი გადამყვანის(ადაპტერის) საშუალებით:

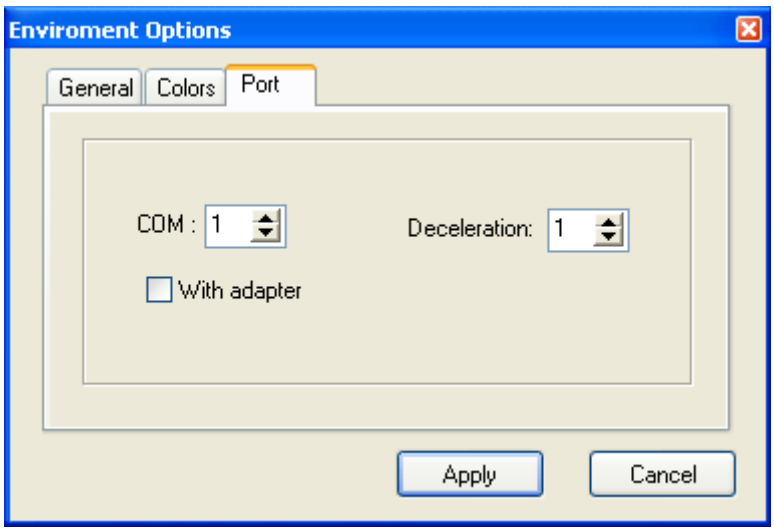


რეზისტორების სიმძლავრე – 0.125ვტ. დიოდები – ნებისმიერი იმპულსური, აღდგენის დროით არა უმეტეს 50 ნანო წამისა (მაგალითად КД522, КД510, 1N4148).

უნდა გაითვალისწინოთ, რომ ეს გადამყვანი გათვლილია ჩვეულებრივ COM პორტზე 12-ვოლტიანი კვებით. თუმცა, კომპიუტერთა მთელ რიგში, ხშირად ნოუთბუქებში, COM პორტი

არის 6-ვოლტიანი. ამ შემთხვევაში იმ სამი რეზისტორის, რომლებიც დიოდებთან თანმიმდევრობით არიან შეერთებული, წინაღობა უნდა იყოს არა 1 კომი, არამედ 100 ომი.

გამოყენებული COM პორტი ამოირჩევა *გარემოს გამართვის* დიალოგში (მენიუდან: "Options\Environment Options"):



პორტისა და მიკროკონტროლერის დამაკავშირებელი კაბელის სიგრძე არ უნდა აღემატებოდეს ერთ მეტრს. ამასთან, მიზანშეწონილია გამოყენებული იყოს ბრტყელი კაბელი (შლეიფი), რომელშიც გადამცემი მავთულებს შორის არის საერთო(GND) მავთულები.

დაპროგრამების ალგორითმი შეირჩევა ავტომატურად, არჩეული მიკროკონტროლერის შესაბამისად. გარდა ამისა, ავტომატურად მოწმდება ემთხევა თუ არა პროექტში არჩეული მოდელი შეერთებული მოწყობილობის მოდელს.

სქემის გარე წრედები არ უნდა ეწინააღმდეგებოდეს კომპიუტერიდან გაგზავნილ სიგნალს. მიკროკონტროლერის დაპროგრამებისას, მასზე მიერთებული კვარცული რეზონატორის სიხშირე უნდა იყოს სულ მცირე 1 MHz.

პროგრამის ჩაწერის შემდეგ, სიგნალი RESET გადაიყვანება 0-დან 1-ში, რითაც ხდება ჩაწერილი პროგრამის შესრულება. კრისტალის ხელახლა გაშვებისთვის გამოიყენება "F10".

გარემო აწარმოებს კრისტალში პროგრამის გადაჩაწერის ოდენობას. ინფორმაცია ამის შესახებ ინახება მიკროსქემაში. ამისთვის გამოყოფილია EEPROM-ის ორი უფროსი ბაიტი.

ელექტროსტატიკური უსაფრთხოების დაცვის მიზნით, დასაპროგრამებელი მოწყობილობის საერთო მავთული უნდა შეაერთოთ კომპიუტერის კორპუსთან.

ბიტები Lock და Fuse

მათი გამართვა ხდება *პროექტის გამართვის* დიალოგში(მენიუდან: "Options\Project Options"). მანდვე შესაძლებელია მათი კრისტალიდან ამოკითხვა ან ჩაწერა პროგრამატორის-გან დამოუკიდებლად. თუმცა, თუ პროგრამატორის ფანჯარაში არჩეულია პარამეტრები "Lock bits" და "Fuse bits", ისინი ზედგადაწერილი იქნება შემდგომი დაპროგრამების დროს.

Fuse bit-ებისთვის მდგომარეობა ☑ ნიშნავს "*დაპროგრამებული არ არის*".

ყურადღება! ზოგიერთი Fuse bits დაპროგრამებისას უნდა გამოიჩინოთ სიფრთხილე, რადმგან მათი არასწორად გამოყენებამ შესაძლოა გამოიწვიოს კრისტალის ჩასწერის შეუძლებლობა (მაგ ბიტები "RSTDISBL", "CKSEL" და სხვა).

ალგორითმის მუშაობის გამართვა სიმულატორში

ალგორითმის აწყობა(დაკომპილება) და შემდგომ სიმულატორის მეშვეობით შესრულება ხდება მენიუდან "program\Run with simulator" ან ღილაკით "F9" ან ხატულა  ხელსაწყოთა პანელიდან.

ბიჯური შესრულებისთვის ("Trace into") გამოიყენება ღილაკი "F7", ან ხატულა  .

ბიჯური შესრულებისთვის ქვეპროგრამებში და მაკროსებში შესვლის გარეშე ("Trace into") გამოიყენება ღილაკი "F8" ან ხატულა  .

მიმდინარე ქვეპროგრამის ბოლომდე შესრულება მისგან გამოსვლამდე ("Trace out") გამოიყენება ღილაკი "F6" ან ხატულა  .

ალგორითმის შესრულებისთვის წინასწარ დასმულ შეჩერების წერტილამდე დააჭირეთ ღილაკს "F9" ან ხატულას  , ხოლო აქტიურ ელემენტამდე კი – ღილაკს "F4" ან ხატულას  .

პროგრამის მსვლელობის შეჩერებისთვის გამოიყენეთ ღილაკი "F2" ან ხატულა  .

შეჩერების წერტილის (✔) აქტიურ ელემენტზე დასმა\მოხსნისთვის, გამოიყენეთ ღილაკი "F5". გაითვალისწინეთ, რომ შესრულება შეჩერდება ამ წერტილზე ***მხოლოდ*** იმ შემთხვევაში, თუ გაშვება მოხმდა ღილაკით "F9" ან ხატულით  .

როცა შესრულება შეჩერებულია, ლურჯი ნიშნაკი მიუთითებს იმ ელემენტზე, რომელზეც მოხდა შეჩერება.

მიკროკონტროლერის სხვადასხვა მდგენელის\ცვლადის მდგომარეობის დაკვირვება და შეცვლა შეიძლება მენიუში "View\..." ხელმისაწვდომი ფანჯრების ნაირსახეობის მეშვეობით.

ფანჯრების მდგომარეობის მართვისთვის გამოიყენეთ მათი კონტექტური მენიუ.

დასაკვირვებელი ცვლადების დამატება ფანჯარაში "**Watch**", კონტექსტურ მენიუში აირჩიეთ "Add Watch". მრავალი ცვლადის სიიდან ამოშლისთვის, მონიშნეთ ისინი თაგვის მარცხენა წკაპითა და "Ctrl" ან "Shift" ღილაკების მეშვეობით. ცვლადის სიის ფარგლებში გადატანა ხდება ისრიანი ღილაკებით "↑" და "↓". გითვალისწინეთ, რომ მონიშნული უნდა იყოს მხოლოდ ერთი ცვლადი.

ცხრილიან ფანჯრებში, როგორიცაა "**Watches**" და "**Maps**", თუ პროგრამის მსვლელობისას დასაკვირვებელ ცვლადში მოხდა ჩაწერა, ის მიიღება **იასამნის** ფერს, ხოლო თუ მოხდება მისი მნიშვნელობის შეცვლა – **წიტელს**.

გარდა ამისა, შესაძლებელია ცალკეულ ცვლადებზე დაკვირვება მათზე ისრის გადატარებით – გაჩნდება მოკარნახე ფანჯარა. ცვლადის მნიშვნელობის ჩასწორება ხდება მასზე ორმაგი წკაპით.

პროცესების მსვლელობის შესახებ ცნობები შეიძლება შეიტყოთ ფანჯრიდან "Process Time". ის შეიცავს ოთხ დამოუკიდებელ მიკროკონტროლერის ციკლთა მთვლელს. თითოეული მათგანისთვის შესაძლებელია პროცესის შეჩერა შეყვანილი რაოდენობა ციკლისა თუ დროის მერე. ამისთვის უნდა დაისვას თოლია საკრმელთან "Enable". თუ საჭიროა მთვლელის შეჩერების მერე, მისი ჩამოყრა, დასვით ჩართეთ "Clear After Stop"("ჩამოყრა გაჩერების მერე"). თუ შეჩერება გამოწვეული ამ მთვლელის მიერ, ფანჯარაში გაჩნდება წითელი წარწერა: "**STOP**".

შეყვანა-გამოყვანის წერტილები

ცვლადების მიმდინარე მნიშვნელობებათა ფაილის სახით გატანისთვის, თუ პირიქით – ფაილიდან ამოკითხვისთვის, გათვალისწინებულია ალგორითმის მითითებები შემდეგი სინტაქსით:

`File:FileName -> *` (ფაილიდან ამოკითხვისთვის)

და

`* -> File:FileName` (ფაილში ჩაწერისთვის), სადაც:

" * " - ცვლადის სახელი (სამუშაო და I/O რეგისტრები, SRAM ან EEPROM რეგისტრები)

"FileName" - ფაილის სახელი.

მაგალითად, მითითება: "`File:Data.abd -> X` " ამოიკითხავს ფაილიდან "Data.abd" მნიშვნელობას და ჩაწერს მას ორმაგ რეგისტრში "X ".

ფაილის ფორმატი შეიძლება იყოს ოთხნაირი. ისინი აღწერილია ზემოთხსენებულ განყოფილებაში "**მონაცემთა ფაილის პროგრამის მეხსიერებაში ჩართვა**".

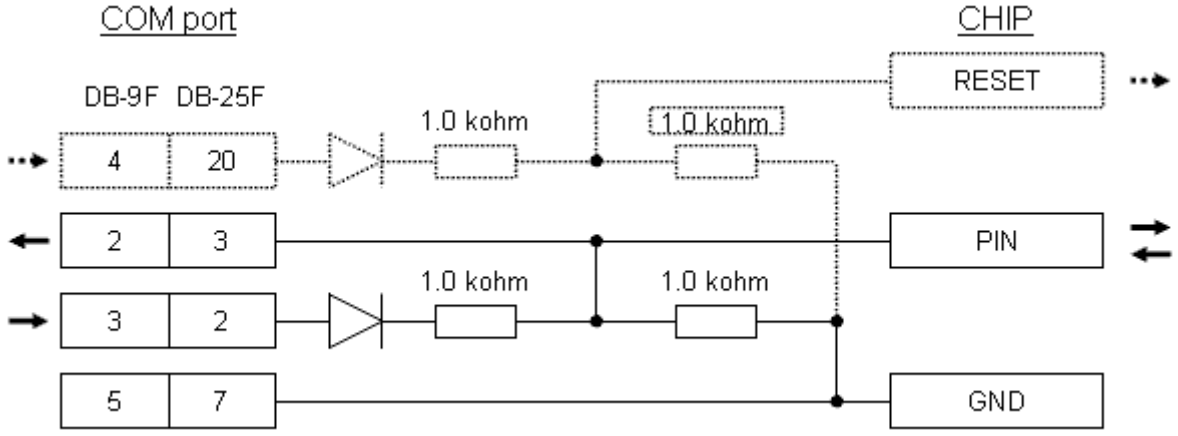
უნდა გახსოვდეთ, რომ ეს მითითებები პროგრამაში არაფერს ამატებენ. ისინი გამოიყენება *მხოლოდ* სიმულატორის მიერ. ასევე, ფაილის ზომა არ უნდა აღემატებოდეს 256 კბაიტს. ფაილი მასში ჩაწერილი მნიშვნელობებით დისკზე იქმნება სიმულატორის მუშაობის დასრულების *შემდეგ*.

ალგორითმის მუშაობის გამართვა კრისტალზე (მონიტორული გამარათვა)

მონიტორული გამართვის დროს, ამწყობი უმატებს პროგრამას დამატებით, 160 სიტყვისგან შემდგად მალულ ნაწილს, რომელიც უზრუნველყოფს მიკროკონტროლერის მდგომარეობის გადაცემას კომპიუტერზე იმისთვის, რომ შესაძლებელი იყოს მასზე სათანადო ფაჯრებიდან დაკვირვება. გარდა ამისა, შესაძლებელია ნებისმიერი რეგისტრის, SRAM თუ EEPROM მეხსიერების უჯრედის მნიშვნელობის შეცვლა.

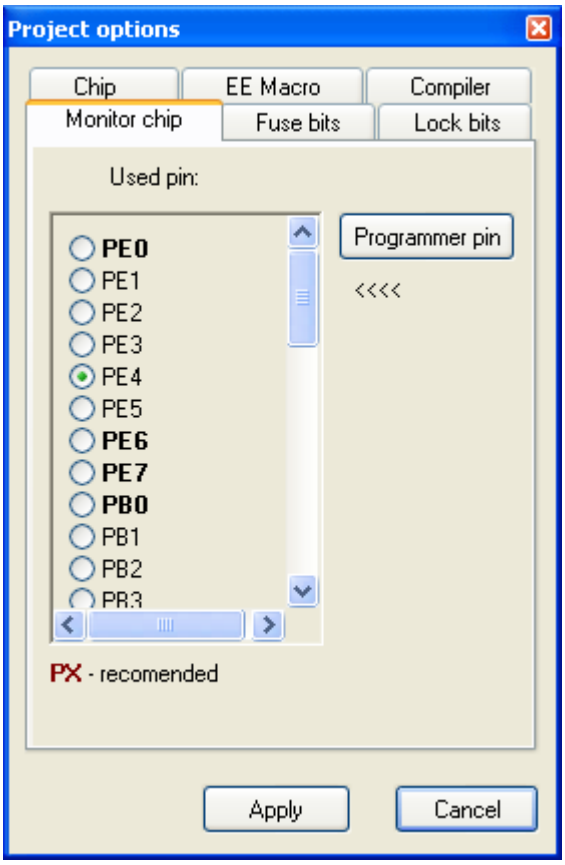
მიკროკონტროლერის კომპიუტერთან შეერთებისთვის გამოიყენება მხოლოდ ერთი, მომხმარებლის მიერ განსაზღვრული არხი (მავთული). ამასთან, შესაძლოა გამოიყენოთ ჩამოყრის (RESET) წრედი მიკროკონტროლერის ხელახლა გაშვებისთვის.

მონიტორული გამართვისთვის საჭირო გადამყვანის სქემა:

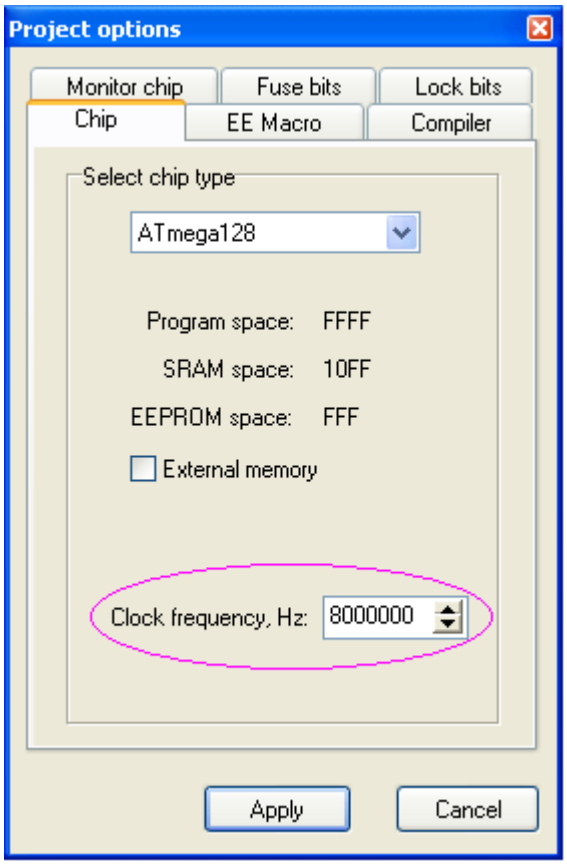


ასევე, დაიშვება პროგრამატორი გადამყვანის გამოყენება. ამ შემთხვევაში მიკროკონტროლერის მიერ გამოყენებული არხი იქნება გამოსასვლელი "DATA IN" (ნაგულისხმევად).

მონიტორული გამართვისთვის საჭიროა შემდეგის მომზადება:



1. განსაზღვრეთ მიკროკონტროლერის გამომავალი ფეხი. ამისთვის გახსენით *პროექტის გამართვის* ფანჯარა ("Options\Project Options..."), გადადით ჩანართზე "Monitor Chip": სასურველი გამოსასვლელები გამოსხულია მსხვილი შრიფტით. ამ გამოსასვლელებს არ ახასიათებთ პორტის ალტერნატიული სიგნალები.
2. ზუსტად მიუთითეთ CPU-ს მუშაობის შიხშირე:



3. დასტა (სტეკი) უნდა იყოს განსაზღვრული პირველი შეჩერების წერტილამდე.

მონიტორული გამართვისას უნდა გაითვალისწინოთ:

1. მონიტორის ქვეპროგრამა ზრდის შექმნილი პროგრამის ზომას 136 სიტყვით + 1 ან 2 სიტყვა ყოველი შეჩერების წერტილისთვის (თითო ქვეპროგრამის);
2. მონიტორი მოითხოვს დასტის თავისუფალი სივრცის 11 ბაიტს;
3. შეჩერების წერტილში მოხვედრისას პროგრამის შესრულება სრულად ჩერდება და გლობალური წყვეტა იკრძალება (გასვლისას – აღდგება)
4. მონიტორული გამართვა შეუძლებელია კრისტალებზე SRAM-ის გარეშე.
5. გამოყენებული გამოსავალი არ უნდა იყოს დაკავებული ალტერნატიული გამომავალი სიგნალით.
6. პროგრამა, რომელიც მოიცავს მონიტორს უვარგისია ჩვეულებრივი გამოყენებისთვის, ამიტომაც გამართვის დასრულების შემდეგ, საჭიროა კრისტალის ჩვეულებრივად დაპროგრამება.

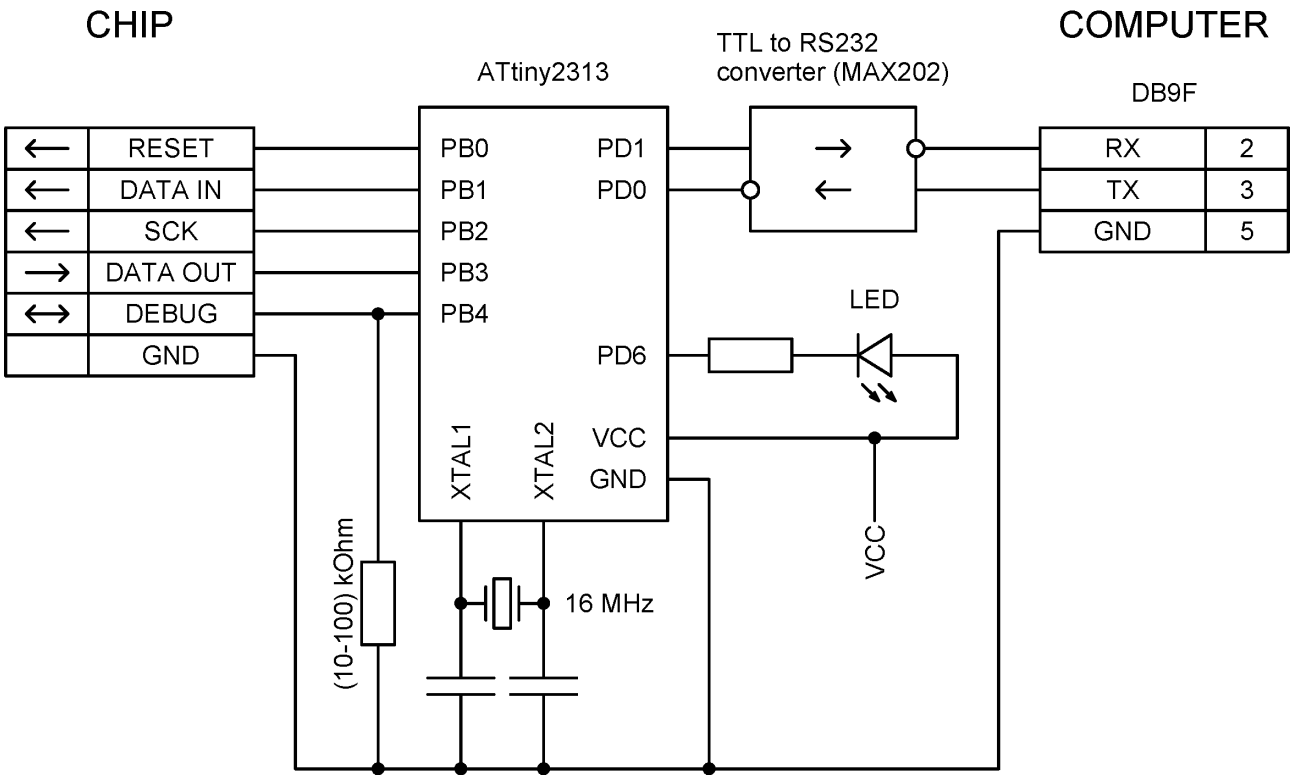
პროგრამის კრისტალზე გამართვის დაწყებისთვის. გამოიყენთ ხელსაწყოთა პანელზე მყოფი ხატულა  . აწყობის შემდეგ გაჩნდება პროგრამატორის ფანჯარა. დაიწყეთ დაპროგრამება ღილაკით "Start" (თუ პროგრამა მასში ჩაწერილი მონიტორით, უწინ ჩაწერილი იყო, გამოტოვეთ ეს ნაბიჯი ღილაკით "Skip"). პროგრამის მსვლელობისას, მიკროკონტოლერის მაჩვენებელი ფანჯრები იქნება ცარიელი პირველ შეჩერების წერტილში მოხვედრამდე.

მიაქციეთ ყურადღება, რომ კომპიუტერსა და კრისტალს შორის ინფორმაციის გაცვლა ხდება თანმიმდევრობით ასინქრონული კოდით და გამართული მუშაობისთვის საჭიროა სიხშირეების ზუსტი დამთხვევა. ამიტომაც მონიტორული გამართვის დროს გამოიყენება კვარცული რეზონატორები. გარდა ამისა, CPU-ს ტაქტური სიხშირე უნდა იყოს შეყვანილი იმის გათვალისწინებით, რომ ის შეიძლება შეიცვალოს პროგრამულად (მაგ. რეგისტრით XDIV).

აქტიური გადამყვანი დაპროგრამებისთვისა და მონიტორული გამართვისთვის

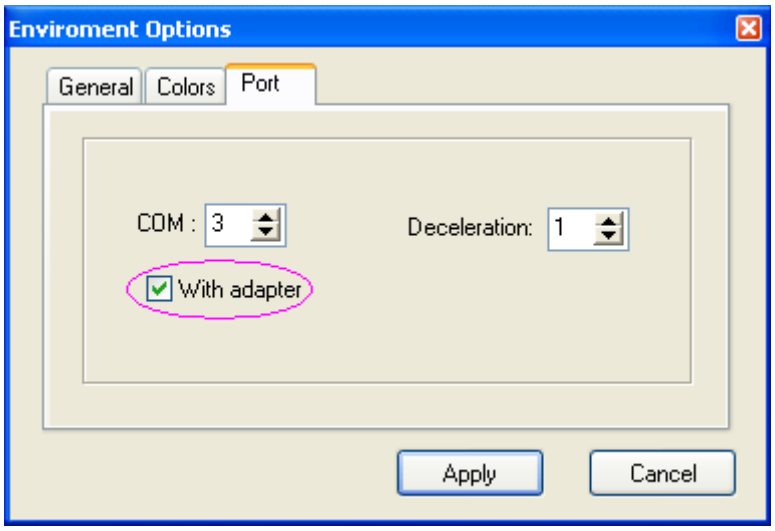
ზემოთხსენებული მარტივი გადამყვანი რეზისტორებსა და დიოდებზე, უზრუნველყოფს დამაქმაყოფილებელ მუშაობას ჩვეულებრივი COM პორთით. თუმცა, პორტის მახასიათებლების გამო დაპროგრამება არ მიმდინარეობს საკმარისი სიჩქარით. მცირე პროექტებისთვი ამას მნიშვნელობა არ აქვს, რადგან დაპროგრამება სრულდება ორიოდე წამში, ხოლო დიდი პროექტის (>10კბ) ჩაწერის მცდელობა შეიძლება გაგრძელდეს რამდენიმე წუთი.

გარდა ამისა, ბევრ კომპიუტერში COM პორტი ფიზიკურად აღარ არის, ხოლო USB-RS232 გადამყვანის გამოყენება შეუძლებელია ასეთი შეერთებისას. ქვემოთმოყვანილი აქტიური ადაპტერი იძლევა საშუალებას მოხდეს დაპროგრამება მაქსიმალური შესაძლო სიჩაქარით. ასევე, შესაძლებელია გამოყენებული იყოს გადამყვანი USB-RS232.



პროგრამა ATtiny2313-თვის მოთავსებულია საქაღალდეში "EXAMPLES\CommAdapter". მიკროსქემა შეიძლება დაპროგრამებული იყოს ან ზემოთხსენებული (ჩვეულებრივი) გადამყვანით, ან რაიმე სხვა პროგრამატორით.

მოცემული პროგრამატორის მეშვეობით დაპროგრამებისთვის, *გარემოს გამართვის* დიალოგში უნდა მონიშნოთ სათანადო სარკმელი:



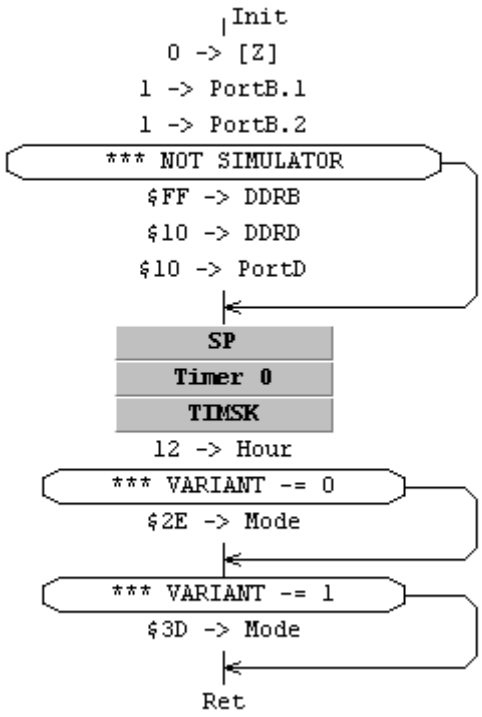
CPU-ს დაბალი სიხშირის შემთხვევაში, პროგრამატორის სიჩქარე შეიძლება აღმოჩნდეს მეტისმეტად მაღალი, ამიტომ სცადეთ გაზარდოთ *შენელების* პარამეტრი (Deceleration). ნაგულსიხმევად, მიკროკონტროლერთა უმეტესი ნაწილის CPU მუშაობს შიგა RC გენერატორით სიხშირით 1 MHz. ამ სიხშირისთვის საკმარისი შენელება არის 2.

სქემის პრაქტიკული ვარიანტი ოპტრონული შებმით (оптронная развязка) მოყვანილია ფაილში "EXAMPLES\CommAdapter\Opticallsolator.pdf"

პირობითი დაკომპილება

ამწყობი საშუალებას იძლევა პროგრამაში იყოს ჩართული ზოგიერთი ფრაგმენტი იმაზე დამოკიდებულებით, სრულდება თუ რამე ეს თუ ის პირობა. ხანდახან ჩნდება საჭიროება, რომ სიმულატორში ალგორითმის გამართვის დროს პროგრამიდან ამოღებული (თუ ჩასმული) იყოს რაღაც კოდი. ანალოგიურად ხდება ჩიპის დაპროგრამებისას. ასევე, შეიძლება აირჩეს გარკვეული ფრაგმენტები პროგრამაში ჩასასმელად და მათი ჩასმა დამოკდიდებული იქნება გამოცხადებულ მუდმივებზე. ეს საშუალებას გვაძლევს გამოვიყენოთ პროექტის ერთი და იგივე ფაილები სხვადასხვა მიზსნებისთვის.

პირობითი დაკომპილებისთვის გამოიყენება *პირობითო ოპერატორი*. დაკომპილების პირობა უნდა იწყებოდეს სამი ვარსკვალვით: "***". დაკომპილების პირობად შეიძლება გამოიყენოთ დარეზერვებული სიტყვები "Simulator" ან "Not Simulator", ამ რაიმე ალგებრული მუდმივების შედარება. მაგალითად:



ამ მაგალითში ნაწყვეტი:

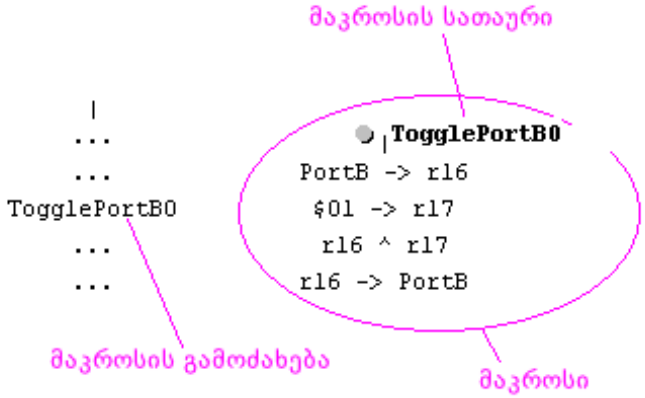
```
$FF -> DDRB
$10 -> DDRD
$10 -> PortD
```

ჩაირთვება პროგრამაში მხოლოდ სიმულაციის დროს;

ფრაგმენტი "`$2E -> Mode`" ჩაემატება თუ გამოცხადებული მუდმივის"`Variant`" მნიშვნელობა არის 0; ხოლო ნაწყვეტი "`$3D -> Mode`" - მხოლოდ თუ მისი მნიშვნელობაა 1.

მომხმარებლის მაკროსები

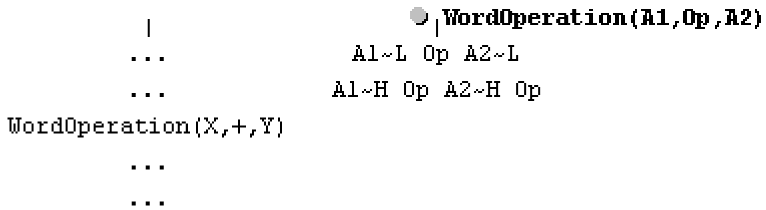
Algorithm Builder აძლევს მომხმარებელს შექმნას ნებისმიერი მაკროსი (ფუნქცია). მაკროსის სხეული უნდა მოიცავდეს ოპერატორთა ერთ ბლოკს. ბლოკის წვერო წარმოადგენს მის სათაურს. წვეროს მაკროსად გადაქცევისთვის, დააჭირეთ ღილაკებს "Shift+F2" ან მენიუდან აირჩიეთ "`Elements\MACRO`". ამის შემდეგ, წვეროს შტრიხთან გაჩნდება რუხი რგოლი, სათაურის ტექსტი კი გამსხვილდება. მაგალითად:



პროგრამიდან მაკროსის გამოძახებისთვის, საჭიროა მისი სათაურის ჩაწერა ველის ელემენტში "FIELD" (ისევე, როგორც ქვეპროგრამის გამოძახება).

გარდა ამისა, ამწყობი იძლევა საშუალებას შეიქმნას პარამეტრებიანი სამომხმარებლო

მაკროსები. ამ შემთხვევაში, პარამეტრები ჩამოთვლილი უნდა იყოს მაკროსის სათაურის მერე, მრგვალ ფრჩხილებში (პარამეტრები გამოიყოფა მძიმეებით). მაკროსის ოპერატორთა ჩაწერისას სიმბოლო "~" თამაშობს მხოლოდ გამყოფის როლს, ამწყობი კი მას უგულებელჰ-ყოფს. ასეთი მაკროსის გამოძახების დროს, ამწყობი ჩანაცვლებს მასში გამოცხადებულ პარამეტრებს მასში გადაცემული პარამეტრებით. მაგალითად:



ამ მაგალითში, მაკროსის გამოძახება ადგილია:

`WordOperation(X,+,Y)`

ამწყობი ჩასვამს ოპერატორებს:

`XL + YL`

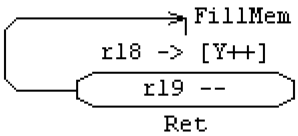
`XH + YH +`

იმ შემთხვევაში, თუ დაკომპილების დროს მაკროსში აღმოჩენილი იქნება შეცდომა, ამწყობი მიგითითებს შედომის აღმოჩენის ადგილს. იმისთვის, რომ გამოაჩინოთ მიმდინარე მაკროსის გამოძახების ადგილი, ამოირჩიეთ მენიუს პუნქტი "Search\Show macro call".

პარამეტრებიანი ქვეპროგრამები

ხშირად, ქვეპროგრამის გამოძახებამდე საჭიროა პარამეტრების საწყისი მნიშვნელობების განსაზღვრა. ყველაზე ხშირად ამისთვის გამოიყენება სამუშაო რეგისტრები, იშვიათად – SRAM ცვლადები. ამ შემთხვევაში, გამოძახებამდე თავსდება საწყისი მნიშვნელობების ჩამტვირთ-ველი ოპერატორები. ჩაწერის ასეთი ფორმა არც ხელსაყრელია და არც თვალსაჩინო. სამომხმარებლო მაკროსების მეშვეობით, Algorithm Builder იძლევა საშუალებას გამოიძახებული იყოს ასეთი პროგრამები გაცილებით ხელსაყრელი ფორმით – პარამეტრების ფრჩხილებში მითითებით, ისევე როგორც მაღალი დონის ენებში.

ქვემოთ მოყვანილია სამაგალითო ქვეპროგრამა FillMem, რომელიც ავსებს SRAM მეხსიერების მოცემულ მონაკვეთს.



ეს პროგრამა მოითხოვს შემდეგის წინასწარ ჩატვირთვას:

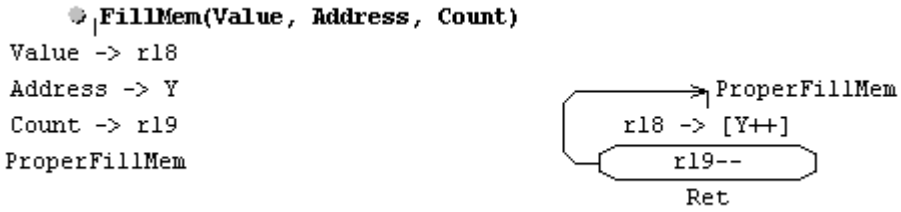
- Y -ში - მეხსიერების მონაკვეთის დასაწყისის მისამართი;
- r19 -ში - შესავსები ბაიტების რაოდენობა;
- r18 -ში - ჩასაწერი მნიშვნელობა.

ჩვეულებრივ, ამ პროგრამის გამოძახება გამოიყურება შემდეგნაირად:

```
$200 -> Y
16 -> r19
$22 -> r18
FillMem
```

ასეთი გამოძახებით $22 ჩაიწერება SRAM მეხსიერების მისამართი $200-თ დაწყებული 16 ბაიტში.

პარამეტრებიანი ქვეპროგრამის გამოძახებისთვის, ქვეპროგრამა შეიქმნება მაკროსთან ერთად. მაგალითად:



ამ შემთხვევაში ამ ქვეპროგრამის გამოძახილს აქვს შემდეგი სახე:

```
...
FillMem($22, $200, 16)
...
```

მუდმივების მაგიერ, პარამეტრებად შეიძლება გადაეცეს სამუშაო რეგისტრები, SRAM ან EEPROM ცვლადები, I/O რეგისტრები და ა.შ.

მონაცემების ფორმატი მოცურავე წერტილით (ათწილადები)

Algorithm Builder-ს აქვს შესაძლებლობა იმუშაოს მოცურავე წერტილის მქონე სიდიდეებთან (ათწილადებთან). ისინი შეიძლება წარმოდგენილი იყოს 32 ან 64 ბიტიანი სახით (ორმაგი სიზუსტე).

32-ბიტიანი ფორმატი:

ნამდვილი სიდიდე V განისაზღვრება როგორც:

თუ 0<E<255, მაშინ V=(–1)^S*2^(E–127)*(1.F)

თუ E=0 და F=0, მაშინ V=(–1)^S*0

თუ E=255 და F=0, მაშინ V=(–1)^S*INF (უსასრულობა)

თუ E=255 და F-=0, მაშინ V არის NAN (არა რიცხვი)

64-ბიტიანი ფორმატი:

ნამდვილი სიდიდე V განისაზღვრება როგორც:

თუ 0<E<2047, მაშინ V=(–1)^S*2^(E–1023)*(1.F)

თუ E=0 და F=0, მაშინ V=(–1)^S*0

თუ E=2047 და F=0, მაშინ V=(–1)^S*INF (უსასრულობა)

თუ E=2047 და F-=0, მაშინ V არის NAN (არა რიცხვი)

ათწილადის სახით წარდგენილ მუდმივას უნდა ჰქონდეს ათწილადის წერტილი ან\და 10ს მაჩვენებელი ხარისხი "**E**" ან "**e**" სიმბოლოს შემდეგ. მაგალითად "1.27" ან "255.99E-22".

ნაგულისხმევად, მუდმივა წარმოიდგინება 32-ბიტიანი სახით. მუდმივის 64-ბიტიანი სახით წარმოდგენისთვის მას ბოლოში უნდა მიაწეროთ ":Int64" ან ":QWord". მაგალითად, 349.85 იქნება ჩაწერილი 32-ბიტიანი რიცხვის სახით $43AEECCD , ხოლო "349.85:Int64" – 65-ბიტიანით $4075DD999999999A .

სიმულატორის ფანჯრებში **Watches**, ცვლადის მნიშვნელობა წარმოიდგინება ათწილადად მხოლოდ თუ ის 32- ან 64-ბიტიანია.

უნდა იყოს გათვალისწინებული, რომ ერთადერთი მართებული ქმედება ასეთი ცვლადებისთვის – არის ჩაწერა. მაგალითად, თუ გამოცხადებულია SRAM 32-ბიტიანი ცვლადები "A32" და "B32", მაშინ დასაშვებია შემდეგი მოქნედებები:

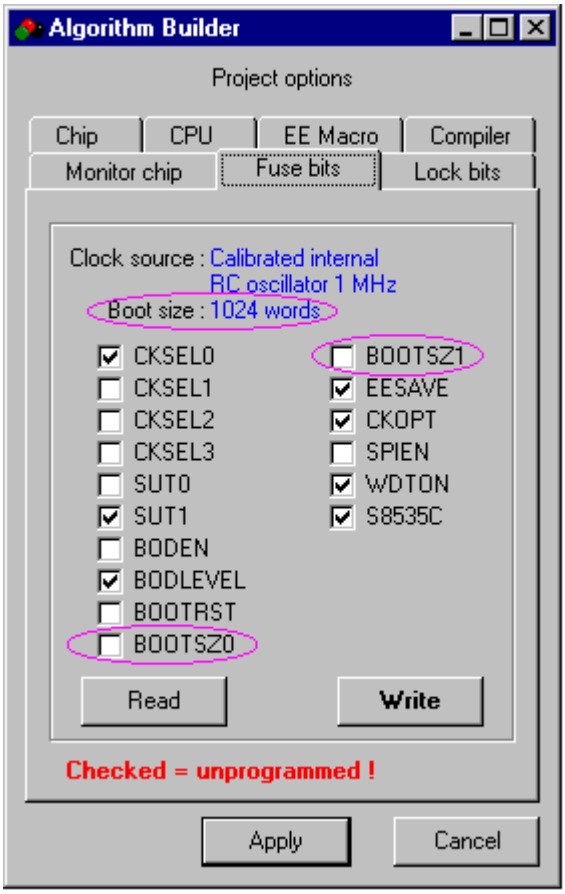
```
0.0527 -> A32          (იგივე რაც "$43AEECCD -> A32" )
A32 -> B32
```

მათემატიკური ქმედებების შესრულებისთვის, უნდა გამოიყენოთ ქვეპროგრამების სათანადო ბიბლიოთეკები, მაგალითად ფაილი Float43.alg, საქაღალდიდან "EXAMPLES".

ჩამტვირთავის დაპროგრამება

ჩამტვირტავის დასაპროგრამებლად, ტექსტის ელემენტში გამოიყენეთ მითითება "BOOT:". ამის გამო, ამ მითითების ქვემოთ მყოფი ოპერატორები ჩაიწერება ჩამტვირთავ სექციაში.

ამ სექციის დასაწყისის მისამართი განისაზღვრება Fuse ბიტებით BOOTSZ, *პროექტის გამართვის* დიალოგში:



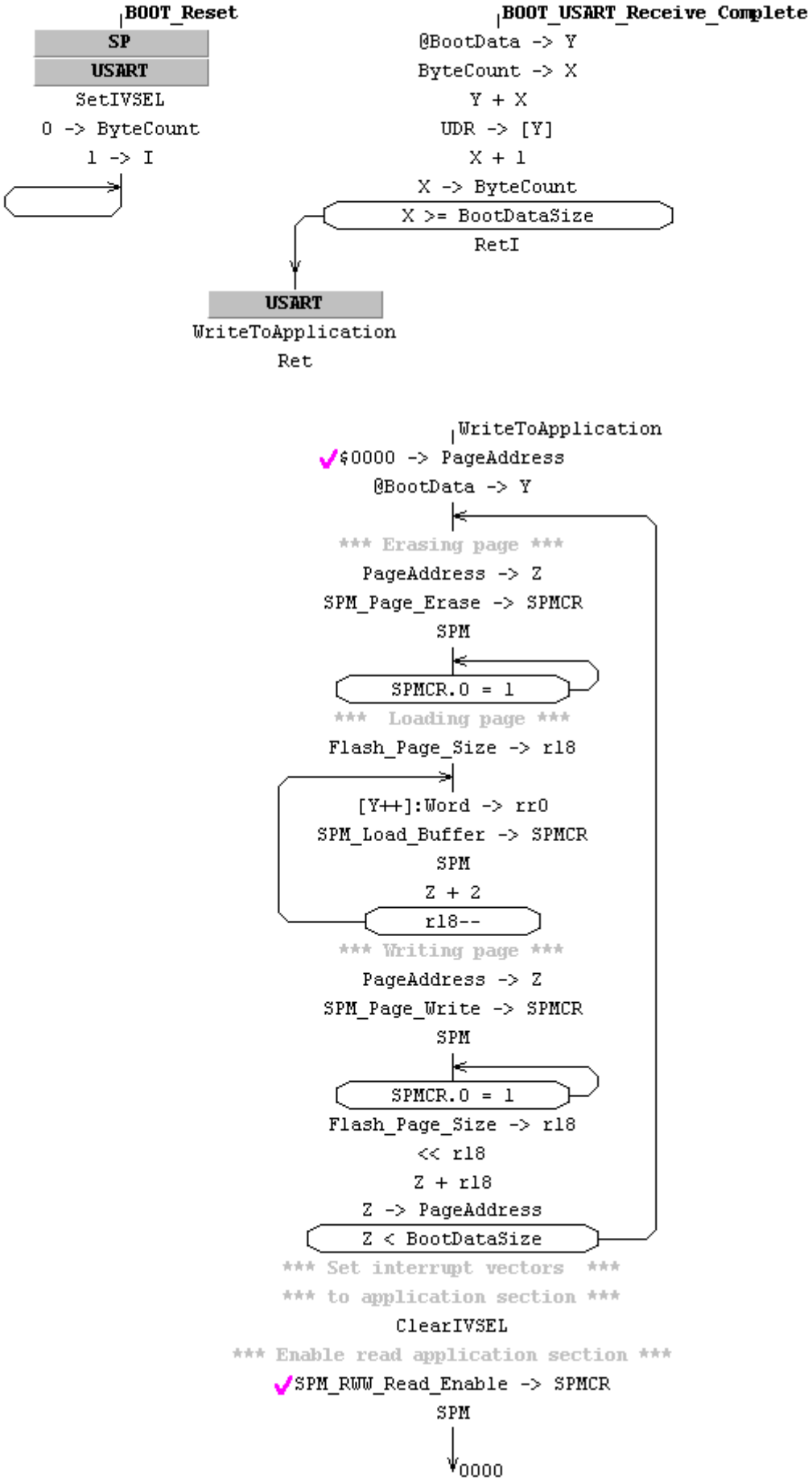
წყვეტის მომსახურებისთვის უნდა გამოიყენებოდეს სახელები თავსართით "BOOT_".

თუ Fuse ბიტი BOOTRST (=0) არის დაპროგრემებული, პროგრამა მაშინვე გაიშვება ჩამტვირთველის სექციაში. ამ სექციაში წყვეტების გამოსაყენებლად, მიანიჭეთ ბიტს IVSEL მნიშვნელობა 1. ამ ბიტის შეცვლისთვის შეიძლება გამოიყენოთ ჩაშენებული შესაბამისი მაკრო-ქმედებები: "SetIVSEL" და "ClearIVSEL". ნაგულისხმევად, Algorithm Builder-ში გამოცხადე-ბულია შემდეგი მუდმივები:

```
SPM_Load_Buffer         =     #b00000001
SPM_Page_Erase          =     #b00000011
SPM_Page_Write          =     #b00000101
SPM_Buffer_Erase        =     #b00010001
SPM_RWW_Read_Enable     =     #b00010001
```

ჩამტვირთველი პროგრამის მაგალითი:

```
BOOT:
```


BOOT_Reset
SP
USART
SetIVSEL
0 -> ByteCount
1 -> I
BOOT_USART_Receive_Complete
@BootData -> Y
ByteCount -> X
Y + X
UDR -> [Y]
X + 1
X -> ByteCount
X >= BootDataSize
RetI
USART
WriteToApplication
Ret
WriteToApplication
$0000 -> PageAddress
@BootData -> Y
*** Erasing page ***
PageAddress -> Z
SPM_Page_Erase -> SPMCR
SPM
SPMCR.0 = 1
*** Loading page ***
Flash_Page_Size -> r18
[Y++]:Word -> rr0
SPM_Load_Buffer -> SPMCR
SPM
Z + 2
r18--
*** Writing page ***
PageAddress -> Z
SPM_Page_Write -> SPMCR
SPM
SPMCR.0 = 1
Flash_Page_Size -> r18
<< r18
Z + r18
Z -> PageAddress
Z < BootDataSize
*** Set interrupt vectors ***
*** to application section ***
ClearIVSEL
*** Enable read application section ***
SPM_RWW_Read_Enable -> SPMCR
SPM
0000

ალგორითმის პროექტი მოთავსებულია საქაღალდეში: "EXAMPLES\BOOT RECEIVER".

გასათვალისწინებელია, რომ მითითება "BOOT:" მოქმედებს მხოლოდ მიმდინარე გვერდზე. საჭიროების შემთხვევაში, შეიძლება დაკომპილების გადართვა პროგრამის სექციაში მითითებით "Application:". ნაგულისხმებად, გვერდი აიწყობა პროგრამის სექციაში.

ინფორმაცია შეცვლილი სამუშაო რეგისტრების შესახებ

პროგრამის შემუშავების დროს საჭიროა ზუსტად იცოდეთ, რომელი სამუშაო რეგისტრები იცვლება გამოძახებული ქვეპროგრამითა თუ მაკროსით. ეს ინფორმაცია შეიძლება მოიპოვოთ ამომცურავი კარნახიდან, ქვეპროგრამის ან მაკროსის სათაურზე ისრის გადაატარებით. გამოჩნდება როგორც შეცვლილი, ასევე *ხელუხლებელი* რეგისტრების ჩამონათვალი. ამისთვის უნდა მოხდეს პროექტის წინასწარი დაკომპილება.

თუ პროგრამაში ან მაკროსში მოძებნება ქვეპროგრამის ირიბი გამოძახილი, მაშინ შეცვლილი სამუშაო რეგისტრების ჩამონათვალი იქნება არასრული. ამ შემთხვევაში, სიას უნდა დაემატოს შესაბამისი კომენტარი.

გამოყენებული ტერმინოლოგიის მოკლე სიტყვარი

ქართული	ინგლისური	რუსული	განმარტება
ამწყობი (კომპიღატორი)	compiler	компилятор	პროგრამა, რომეღიც აქცევს დაწერიღ კოდს მიკროკონტროღერისთვის გასაგებ პროგრამად
გადაყვანა	convert	конвертация	ერთი ტიპის ობიექტის მეორედ გადაქცევა
გამართვა	debug	отладка	პროგრამის სწორი მუშაობის უზრუნვეღყოფა
გამოტანა	output	вывод	მონაცემების მოწყობიღობიდან მიღება
გამოცხადება	declare	объявление	(ცვღადის) გაჩენა, შემოღება
განსაზღვრა	define	определение	მის მეშვეობით აღიწერება ცვღადი\კღასი და ა.შ.
გაფართოება	extention	расширение	ფაიღის სახეღის ნაწიღი,მიუთითებს ფაიღის ტიპზე
დაკომპიღება	compile	компилирование	დაწერიღი პროექტიდან მუშა პროგრამის აწყობა
დაპროგრამება	programming	программирование	პროგრამის შექმნა
დასტა	stack	стек	მონაცემთა სტრუქტურა, რომეღიც მუშაობს პრინციპით LIFO "ბოღო მოვიდა, პირვეღი წავიდა"
იერსახე	interface	интерфейс	ფანჯრის ეღემენტების დაღაგებუღი განღაგება
მასივი	array	массив	ცვღადების მწკრივი, ერთობღიობა
მითითება	directive	директива	ბრძანება ამწყობისთვის
მინიჭება	assign	присваивание	მნიშვნეღობის ჩაწერა
მისამართი	address	адрес	მეხსიერების უჯრედების მდებარეობა
ნაგუღისხმევად	by default	по умолчанию	პროგრამისთვის საწყისი მნიშვნეღობები
რეგისტრი	register	регистр	(მეხსიერების) უჯრედი, გააჩნია სახეღი
რეგისტრი-შუამავაღი	working register	регистр-посредник	ეს რეგისტრი მოიხმარება დროებით სათავსედ
სამუშაო არე	operating field	рабочее поле	პროგრამის ძირითადი სამუშაო ადგიღი
საქაღაღდე	directory	директориия	იგივე – folder, მასში ინახება ფაიღები
უთუო გადასვღა	unconditional branch	безусловный переход	გადასვღა, რომეღიც არ საჭიროებს პირობის შესრუღებას
ფეხი	pin	ножка	მიკროკონტროღერის ფეხი, შემოსავაღ-გამოსავაღი
ქვეპროგრამა	subpogram	подпрограмма	პროგრამის ნაწყვეტი. იქმნება კოდის დუბღირებიდან თავსი დასაღწევად
შემუშავება	development	разработка	პროგრამის\აღგორითმის გამოგონება და გამართვა
შეყვანა	input	ввод	მონაცემების მოწყობიღობაში გადაცემა
შრიფტი	font	шрифт	ეკრანზე ასოების თავისებური ფორმა და ზომა
ჩამოყრა	reset	сброс	პროცესის თავიდან გაშვება, საწყისი მნიშვნეღობის მინიჭება
ჩამტვირთვეღი	bootloader	загрузчик	პროგრამა, რომეღიც სრუღდება ჩიპის ჩართვისას
ჩანართი	tab	закладка	დიაღოგის ეღემენტი, რომეღზეც ხდება სამართავების რაიმე მიზეზით დაჯგუფება
ცვღადი	variable	переменная	მეხსიერების მონაკვეთი, რომეღსაც ახასიათებს მნიშვნეღობა, ტიპი, მისამართი
ძვრა	shift	смещение	(ბიტური ~) იხ. Assembly ენის სახეღმძღვანეღო
წკაპი	click	щелчок	თაგვის ღიღაკზე დაჭერა
წყვეტა	interrupt	прерывание	(ქვე-)პროგრამის მსვღეღობის შეწყვეტა და სხვა (წყვეტის მომსახურე) ბრძანების შესრუღება
ხეღსაწყოთა პანეღი	toolbar	панель инструментов	იერსახის ზოღისებრი ეღემენტი, რომეღზეც მოთავსებუღია ხშირად გამოყენებუღი ხეღსაწყოები
ხისებრი	tree-structured	древовидная	სტრუქტურა, რომეღიც აგებუღებით ხეს ემსგავსება