

Algorithm Builder for AVR

by Gennady Gromov, Tula Telecom

The most commonly used programming tools for AVR Microcontrollers are a C language compiler or assembler. Each approach has relative strengths and limitations. Assembly language programs typically produce smaller code size but there is an overhead associated with programming in Assembly language. Issues associated with programming in Assembly are non-visual listing and redundant detailing. Therefore, software developed in assembly language is time consuming. Programmers demand more efficient tools. C language for the 8-bit AVR Microcontroller is gaining in popularity because of code maintainability, portability and programming speed. When C language is used, the programmer has a visual listing and is released from redundant detailing. This appreciably accelerates code development. There is a penalty associated with C Language programming: less efficient code. There is a solution the algorithm tool for the AVR. Algorithm builder fits between the Assembler and C compiler. Code is still written in Assembly but the improved graphical user interface development time can be reduced 3-5 X and code size reduced 2 to 3X. This provides benefits of both approaches. Development time is similar to C language programming while code density is improved. This is a result of the graphical user interface, which allows use of flow charts and tree like branch structures. Additionally the instruction notation form was simplified and the mnemonics of the Assembler was changed to visual notation:

in place of:	use's:
MOV R0,R1	R1 -> R0
LDI R20,\$5E	\$5E -> r20
ADD R0,R1	R0 + R1
ANDI R20,25	R20 & 25
SBIW X,15	X + 15
LSR R7	R7 >>
SBI PortB,3	1 -> PortB.3

The large instructions set, such as MOV, LD, LDI, LDD, ST, STD, IN, OUT, CBI, SBI etc., are changed to only one notation: "->". Using this notation is more convenient than mnemonics, and the operation matter is easy to understand, but the main in AB is a graphic interface. The main problem of classical assembler is limitations of the text editor. The written program is built as a continuous vertical line. The logical structure of program is hidden and can be built only in a programmer's imagination or on a paper.

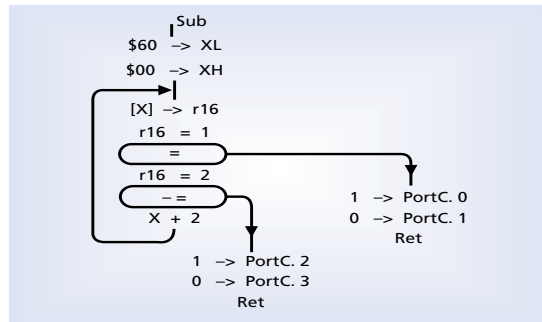
Any program can be divided into the some logical accomplished fragments. These fragments (blocks) are terminated by operations after which, the linear program execution is unambiguously ended. There are subroutine return ("RET", "RETI") and unconditional branches. Below is example of subprogram containing three such blocks:

```

Sub:  ldi  XL,$60
      ldi  XH,$00
MO:   ld   r16,X
      cpi  r16,1
      breq M1
      cpi  r16,2
      brne M2
      sbiw X,2
      rjmp M0
M1:   sbi  PortC,0
      cbi  PortC,1
      ret
M2:   sbi  PortC,2
      cbi  PortC,3
      ret
    
```

The first block is begins from the "Sub" label and terminated by unconditional branch ("jmp"). The second and third blocks are begin from the "M1" and "M2" labels and terminate at the subroutine return operators ("ret").

The "Algorithm Builder" allows visual segregation of these blocks and places them on a plane. In result, the logical structure of program becomes visual.



The above subroutine example will be looked in the following way in AB: Five base elements are used in Algorithm Builder for the building of program construction. There are:

- LABEL,
- VERTEX,
- FIELD,
- CONDITIONAL BRANCH and
- UNCONDITIONAL BRANCH.

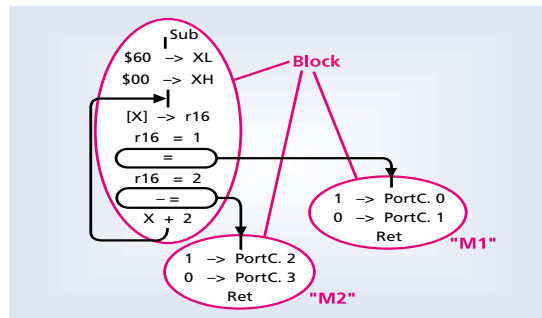
The "LABEL" element is represented as a short vertical line in the block. Label can have an optional name at left or right side of the line. The label purpose is like to label purpose in a classical assembler. It is a marking of places in the program for the branches.

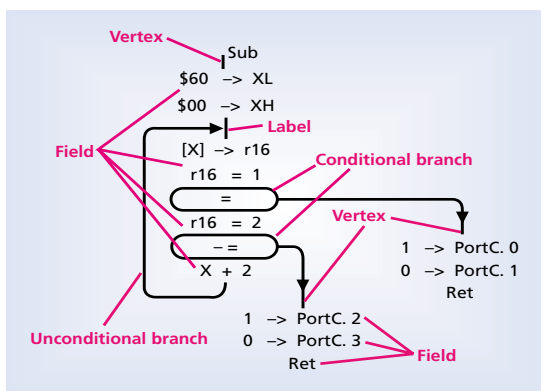
The image and purpose of "VERTEX" is similar to "LABEL", but the vertex is always begins a block and determines this geometrical placing.

The "FIELD" element is most used. It is represented by the in-block centered string. The "FIELD" is used for writing down the most operations of microcontroller except the branches.

The image of CONDITIONAL BRANCH is an oval with an entered branch condition and a vector as an outgoing line with an arrow at the end. The end of line should terminate at LABEL or VERTEX to which the branch will be carried out when the entered condition is fulfilled.

The image of UNCONDITIONAL BRANCH is an outgoing from the block axis line with an arrow at the end similar to CONDITIONAL BRANCH vector.

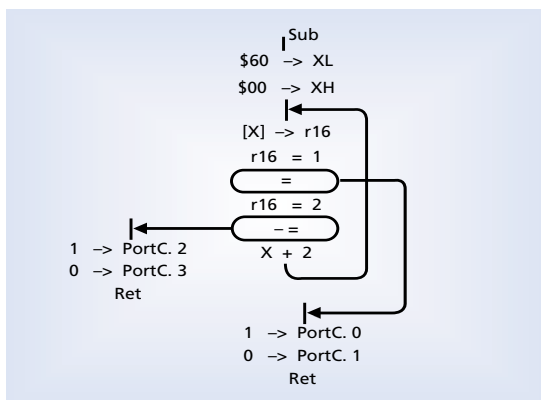




Note the example of algorithm in AB contains only one label name: "Sub", and the "M0", "M1", and "M2" names are absent. In the classical assembler the branch addressing is carried out by labels' names. The use on multiple branch and label names result in a listing crammed with vast number of labels names which are inevitable ballast.

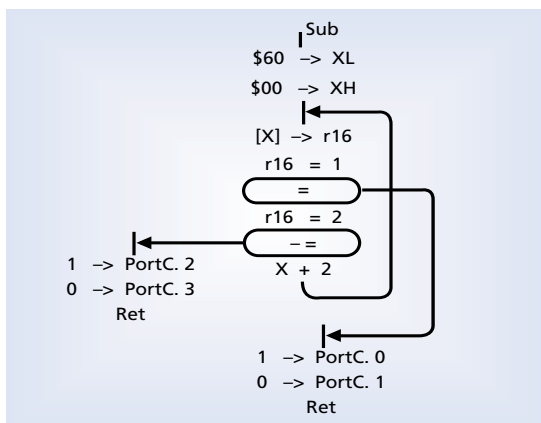
In AB the branches are defined graphically by vectors. Therefore the necessity of labels' names is absent. This eliminates the names of subroutines' entries only. In above example it is a "Sub".

The editor allows the arbitrary placing of program blocks on a plane according to programmer's design. For example, the above subroutine can be redesigned as below:



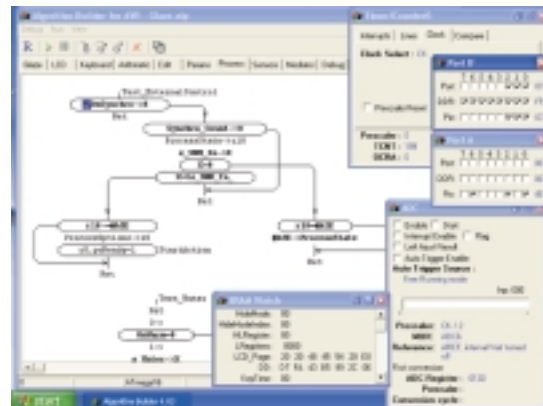
In any case, the compilation result will be equal.

The editor allows ending a vector at segment of another vector, end of which is as shown below:

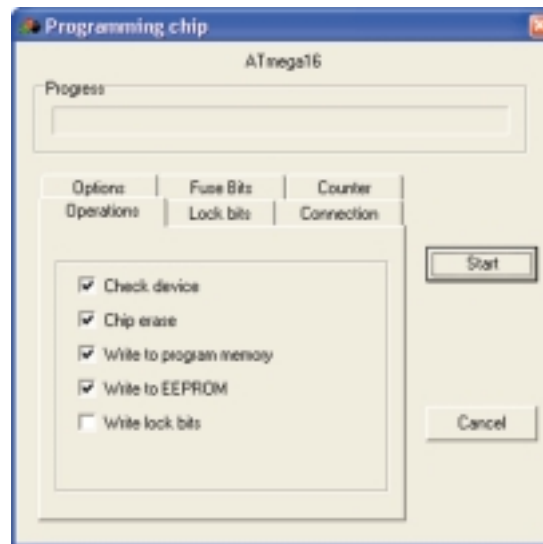


The AB environment is all-sufficient. It contains the editor, compiler, simulator (debugger) and in-circuit serial programmer of a chip. I.e. this tool covers the whole design cycle including programming of a chip. The Capability of the simulator are similar to AVR Studio. In-circuit programming is provided by connecting the chip directly to LPT without additional devices. But the manual contains the recommended scheme of optical isolator.

The opened simulator:



The programmer window:



The Algorithm Builder supports most of AVR chips

The using of AB is very simply. It can be recommended both for beginners and for professional programmers.

This article covers only base aspects of work in AB. This environment allows the design in macro-level with operating of multi-byte values with sign; it supports the special inclusions into the algorithm providing the convenient setting of peripheral components; and others.

You can find the Algorithm Builder on AVR freaks web page (<http://www.avrfreaks.net/>) and at: (<http://home.tula.net/algrom/english.html/>).

The additional possibilities of AB will be stated in a following article in the next issue.