AVR Assembler Examples

AVR specific examples

Credit to Dr. Robucci for slide information

m169pdef.inc

- Include file detailing register and bit definitions

Excerpt:

```
;* When including this file in the assembly program file, all I/O register
;* names and I/O register bit names appearing in the data book can be used.
;* In addition, the six registers forming the three data pointers X, Y and
;* Z have been assigned names XL - ZH. Highest RAM address for Internal
;* SRAM is also defined
;*
;* The Register names are represented by their hexadecimal address.
;*
;* The Register Bit names are represented by their bit number (0-7).
```

```
.equ    DDRB    = 0x04
.equ    PINB    = 0x03
.equ    PORTA   = 0x02
.equ    DDRA    = 0x01
```

```
.equ    PORTB1  = 1     ; Port B Data Register bit 1
.equ    PB1     = 1     ; For compatibility
```

1<<PB5 =
8b'00000001 << 5

allows

```
in      r16,PORTB               ;read PORTB latch
sbr     r16,(1<<PB6)+(1<<PB5)   ;set PB6 and PB5
out     PORTB,r16               ;output to PORTB
```

SBR – Set bits in reg
Equivalent to an ORI
REQUIRES MASKS, not bit number

Timer Example

From m169pdef.inc

```
.equ    TIFR0   = 0x15

; TIFR0 - Timer/Counter0 Interrupt Flag register
.equ    TOV0    = 0     ; Timer/Counter0 Overflow Flag
.equ    OCF0A   = 1     ; Timer/Counter0 Output Compare Flag 0
```

allows

In your program:

```
in    r16,TIFR0          ;read the Timer Interrupt Flag Register
sbrc  r16,TOV0           ;test the overflow flag (use bit#)
rjmp  TOV0_is_set        ;jump if set
 ...                     ;otherwise do something else
```

These allow better readability than the I/O register number

SRAM and Stack Pointer

- This sets the stack pointer to the end of the SRAM
  - HIGH(x) - upper 8 bits of a 16-bit word
  - Low(x) - lower 8 bits of a 16-bit word
  - SPH is stack pointer high
  - SPL is stack pointer low
- Necessary for stack use

```
.equ    SRAM_START      = 0x0100
.equ    SRAM_SIZE       = 1024
.equ    RAMEND  = 0x04ff
```

allows

```
LDI R0, HIGH(RAMEND) ; upper byte
OUT SPH,R0           ;
LDI R0, LOW(RAMEND)  ; lower byte
OUT SPL,R0
```

Resources

- http://www.avr-asm-tutorial.net/avr_en/beginner/index.html

- http://www.avr-asm-tutorial.net/avr_en/beginner/COMMANDS.html
  - Commands sorted by function
- http://www.avr-asm-download.de/beginner_en.pdf
  - Beginner introduction to Assembly of ATMEL-AVR µProcessors

MACROS
- http://www.avrfreaks.net/forum/tutasmmacros-assembler?name=PNphpBB2&file=viewtopic&t=101529
- http://www.avrbeginners.net/assembler/macros.html

SUBROUTINES
- http://www.avr-tutorials.com/assembly/writing-assembly-subroutines-avr-microcontroller
  - Implementing Subrotines

Directives

- .EQU – Set a symbol equal to a constant expression
  - .EQU io_offset = 0x23
- .SET – assign a value to a label
  - Can be redefined later in the program
  - .SET FOO = 0x114
  - ……
  - .SET FOO = FOO+1
- .DEF – Set a symbolic name on a register
  - Can be redefined later in the program
  - .DEF temp=R16

Macro Directive

- .MACRO macroname
  - Tells assembler that this is the start of a macro
- When name is used later in program, definition is expanded at the place it is used
- Can take up to 10 parameters
  - @0-@9
- Terminated by .ENDMACRO

Macro Example

Syntax:

.MACRO macroname

Example:

```
.MACRO SUBI16 ; Start macro definition
subi @1,low(@0) ; Subtract low byte
sbci @2,high(@0) ; Subtract high byte
.ENDMACRO ; End macro definition
.CSEG ; Start code segment
SUBI16 0x1234,r16,r17 ; Sub.0x1234 from r17:r16
```

#define

- Create a pre-processor function-style macro

- Example
  - #define BITMASK(x) (1<<x)
  - #define BITMASK3(x1,x2,x3) (BITMASK(x1)+BITMASK(x2)+BITMASK(x3))

Some built in functions

LOW(expression) returns the low byte of an expression
HIGH(expression) returns the second byte of an expression
BYTE2(expression) is the same function as HIGH
BYTE3(expression) returns the third byte of an expression
BYTE4(expression) returns the 4th byte of an expression
LWRD(expression) returns bits 0-15 of an expression
HWRD(expression) returns bits 16-31 of an expression
PAGE(expression) returns bits 16-21 of an expression
EXP2(expression) returns 2^expression

Some built in functions(cont.)

LOG2(expression) returns the integer part of log2(expression)

INT() Truncates a floating point expression to integer (i.e. discards fractional part)

FRAC() Extracts fractional part of a floating point expression (i.e. discards integer part).

Q7() Converts a fractional floating point expression to a form suitable for the FMUL/FMULS/FMULSU instructions. (sign + 7-bit fraction)

Q15() Converts a fractional floating point to the form returned by the FMUL/FMULS/FMULSY instructions (sign + 15-bit fraction).

ABS() Returns the absolute value of a constant expression.

Stack and Functions

- Using functions requires the stack
- Using the stack requires that it be initialized
  - See slide 4
- After which point the following commands may be used
  - PUSH
  - POP
  - RCALL
  - RET

Function Example

- The program to the right toggles the logic value of pins on portB with a delay after each change

```
.include "m8515def.inc"
        ;Initialize the microcontroller stack pointer
        LDI   R16, low(RAMEND)
        OUT   SPL, R16
        LDI   R16, high(RAMEND)
        OUT   SPH, R16

        ;Configure portB as an output put
        LDI   R16, 0xFF
        OUT   DDRB, R16

        ;Toggle the pins of portB
        LDI   R16, 0xFF
        OUT   PORTB, R16
        RCALL  Delay
        LDI   R16, 0x00
        OUT   PORTB, R16
        RCALL  Delay

        ;Delay subroutine
Delay:     LDI  R17, 0xFF
loop:     DEC  R17
        BRNE loop
        RET
```

Pasted from <http://www.avr-tutorials.com/assembly/writing-assembly-subroutines-avr-microcontroller>

Setting IO

```
//.INCLUDE "m169Pdef.inc"

.ORG 0x00000

//compute memory mapped io address
.EQU io_offset = 0x20
.EQU PORTA_MM = io_offset + PORTA  ; PORTA is 0x02

//set i/o bits using i/o register direct commands(cannot be used w/ ext. i/o regs)
SBI PORTA, 6     //set bit I/O using bit number
CBI PORTA, 6     //clear bit I/O
SBI PORTA, 5     //set bit I/O

//clear bit I/O using indirect access
LDI ZL , low(PORTA_MM)  //load immediate to register
LDI ZH , high(PORTA_MM) //low() and high() byte macros provided automatically
LD R16, Z               //load indirect from memory to reguster R16 using memory
address R31,R30
CBR R16, EXP2(7)        //clear bitS in register requires mask instead of a bit number
ST Z, R16               //store indirect to memory address R31,R30 from reguster R16

//set bit I/O using indirect access
LDI ZL , low(PORTA_MM)
LDI ZH , high(PORTA_MM)
LD R16, Z
SBR R16, EXP2(7)
ST Z, R16

//clear bit I/O using direct (memory) access
LDS R16 , PORTA_MM
CBR R16, EXP2(6)
STS PORTA_MM, R16

//set bit in I/O using direct (memory) access
LDS R16 , PORTA_MM
SBR R16, EXP2(7)
STS PORTA_MM, R16
```

Implementing Delays

Create a loop

```
ldi RTEMP, 255 ; 255 could also be a variable here
the_delay:
dec RTEMP
brne the_delay
```

Use a few nop instructions

```
nop ; 1 clock
nop ; 1 clock
nop ; 1 clock
```

A combination for longer delays

```
ldi RTEMP, 255 ; 255 could also be a variable here
the_delay:
nop
nop
nop
nop
nop
dec RTEMP
brne the_delay
```

Create loops within loops

```
; outer loop
ldi RTEMPB, 255 ; 255 could be a variable so the
                ; inner loop sets the delay step size
outer_delay:
; inner loop
ldi RTEMPA, 122 ;
inner_delay:
nop
nop
nop
nop
nop
dec RTEMPA
brne inner_delay
dec RTEMPB
brne outer_delay
```

Using double-word operations for longer delays.

```
LDI 25, 0x01 //high
LDI 24, 0xFF //low
Loop:
SUBIW R25:R24, 1
BRNE Loop
```

Delays (continued)

- All of the previous delay examples were software defined
  - Dependent upon CPU frequency
- Note on nops (no operations)
  - There may be productive work that can be done with these clock cycles
  - i.e. check status registers, check I/O, etc…
- Precise delays use hardware timers
  - To be learned later

Jumping

- Conditional Jumps
  - SBIC – Skip if bit in I/O register cleared
  - SBIS – Skip if bit in I/O register set
  - SBRC – Skip if bit in register cleared
  - SBRS – Skip if bit in register set
- Unconditional Jumps
  - RJMP k:
    - Program execution continues at PC+k+1
    - 2 clock cycles, smaller address range
  - JMP k:
    - Program execution continues at k
    - 3clock cycles, full address range

Combining jumps

```
.def = JOYSTICK_INPORT = PINB
.equ UP_BUTTON_BIT = 5;
.equ UP_BUTTON_MASK = (1<<UP_BUTTON_BIT);
.equ UP_BUTTON_MASK_CMP = (0xFF - UP_BUTTON_MASK);
.def RTEMP = r16


;skip if bit in register set followed by branch
; branch occurs if button was pressed
sbis JOYSTICK_PORT, UP_BUTTON_BIT
rjmp somewhere
```

In comparison,

```
in RTEMP, JOYSTICK_PORT
andi RTEMP, UP_BUTTON_MASK
brne somewhere
```

Combining Jumps (cont.)

- Example: Wait on bit to change to a 1
  - back_here:
  - SBIS PINB, 0
  - rjmp back_here
  - … (continue)

- How to check or wait on multiple bits?
  - Hint: .equ BUTTON_MASK = (UP_MASK+DOWN_MASK)