COMP2121: Microprocessors and Interfacing

AVR Assembly Programming (II)
Basic AVR Instructions

http://www.cse.unsw.edu.au/~cs2121

Lecturer: Hui Wu

Term 2, 2019

1

1

Contents

- Data transfer instructions
- Bit and bit test instructions
- Sample AVR assembly programs

2

2

Copy Register

- Syntax: mov Rd, Rr
- Operands: Rd, Rr ∈ {r0, r1, …, r31}
- Operation: Rd←Rr
- Flag affected: None
- Encoding: 0010 11rd dddd rrrr
- Words: 1
- Cycles: 1
- Example:

```
mov r1, r0          ; Copy r0 to r1
```

3

3

Copy Register Pair

- Syntax: movw Rd+1:Rd, Rr+1:Rr
- Operands: d, r ∈ {0, 2, …, 28, 30}
- Operation: Rd+1:Rd←Rr+1:Rr
- Flag affected: None
- Encoding: 0000 0001 dddd rrrr
- Words: 1
- Cycles: 1
- Example:

```
movw r21:r20, r1:r0          ; Copy r1:r0 to r21:r20
```

4

4

Load Immediate

- Syntax: ldi Rd, k
- Operands: Rd∈{r16, r17, …, r31} and 0 ≤ k ≤ 255
- Operation: Rd←k
- Flag affected: None
- Encoding: 1110 kkkk dddd kkkk
- Words: 1
- Cycles: 1
- Example:

```
ldi  r16, $42        ; Load $42 to r16
```

5

5

Load Indirect

- Syntax: ld Rd, v
- Operands: Rd∈{r0, r1, …, r31} and v∈{x, x+, -x, y, y+, -y, z, z+, -z}
- Operation:
  (i) Rd←(v) if v ∈{x, y, z}
  (ii) x←x-1 and Rd←(x) if v =-x
  y←y-1 and Rd←(y) if v =-y
  z←z-1 and Rd←(z) if v =-z
  (iii) Rd←(x) and x←x+1 if v =x+
  Rd←(y) and y←y+1 if v =y+
  Rd←(z) and z←z+1 if v =z+
- Flag affected: None
- Encoding: Depends on v. Refer to AVR Instruction Set for details
- Words: 1
- Cycles: 2
- Comments: Post-inc and pre-dec are used to load contiguous data.

6

6

Load Indirect with Displacement

- Syntax: ldd Rd, v
- Operands: Rd∈{r0, r1, …, r31} and v∈{y+q, z+q}
- Operation: Rd←(v)
- Flag affected: None
- Encoding: Depends on v. Refer to AVR Instruction Set for details
- Words: 1
- Cycles: 2
- Example:

```
clr r31          ; Clear Z high byte
ldi r30, $60     ; Set Z low byte to $60
ld r0, Z+        ; Load r0 with data space loc. $60(Z post inc)
ld r1, Z         ; Load r1 with data space loc. $61
ldi r30, $63     ; Set Z low byte to $63
ld r2, Z         ; Load r2 with data space loc. $63
ld r3, -Z        ; Load r3 with data space loc. $62(Z pre dec)
ldd r4, Z+2      ; Load r4 with data space loc. $64
```

- Comments: ldd is used to load an element of a structure.

7

7

Load direct

- Syntax: lds Rd, k
- Operands: Rd∈{r0, r1, …, r31} and 0 ≤ k ≤ 65535
- Operation: Rd←(k)
- Flag affected: None
- Encoding: 1001 000d dddd 0000 kkkk kkkk kkkk kkkk
- Words: 2
- Cycles: 2
- Example:

```
lds r2, $FF00    ; Load r2 with the contents of data space location $FF00
inc r2           ; add r2 by 1
sts $FF00, r2    ; Write back
```

8

8

Store Indirect

- Syntax: st v, Rr
- Operands: Rr∈{r0, r1, …, r31} and v∈{x, x+, -x, y, y+, -y, z, z+, -z}
- Operation: (i) (v)←Rr if v ∈{x, y, z}
  (ii) x←x-1 and (x)←Rr if v =-x
  y←y-1 and (y)←Rr if v =-y
  z←z-1 and (z)←Rr if v =-z
  (iii) (x)←Rr and x←x+1 if v =x+
  (y)←Rr and y←y+1 if v =y+
  (z)←Rr and z←z+1 if v =z+
- Flag affected: None
- Encoding: Depends on v. Refer to AVR Instruction Set for details
- Words: 1
- Cycles: 2
- Comments: Post-inc and pre-dec are used to store contiguous data.

9

9

Store Indirect with Displacement

- Syntax: std v, Rr
- Operands: Rd∈{r0, r1, …, r31} and v∈{y+q, z+q}
- Operation: (v)←Rr
- Flag affected: None
- Encoding: Depends on v. Refer to AVR Instruction Set for details
- Words: 1
- Cycles: 2
- Example:

```
clr     r29          ; Clear Y high byte
ldi     r28, $60     ; Set Y low byte to $60
st      Y+, r0       ; Store r0 in data space loc.  $60(Y post inc)
st      Y, r1        ; Store r1 in data space loc. $61
ldi     r28, $63     ; Set Y low byte to $63
st      Y, r2        ; Store r2 in data space loc. $63
st      -Y, r3       ; Store r3 in data space loc. $62 (Y pre dec)
std     Y+2, r4      ; Store r4 in data space loc. $64
```

- Comments: std is used to store an element of a structure.

10

10

Store direct

- Syntax: sts k, Rr
- Operands: Rr∈{r0, r1, …, r31} and 0≤ k ≤65535
- Operation: (k)←Rr
- Flag affected: None
- Encoding: 1001 001d dddd 0000 kkkk kkkk kkkk kkkk
- Words: 2
- Cycles: 2
- Example:

```
lds r2, $FF00      ; Load r2 with the contents of data space location $FF00
add r2, r1         ; add r1 to r2
sts $FF00, r2      ; Write back
```

11

11

Load Program Memory (1/3)

Syntax:	Operands:	Operations:
(i) LPM	None, R0 implied	R0←(Z)
(ii) LPM Rd, Z	0 ≤ d ≤ 31	Rd←(Z)
(iii) LPM Rd, Z+	0 ≤ d ≤ 31	Rd←(Z)

- Flag affected: None
- Encoding: (i) 1001 0101 1100 1000
  (ii) 1001 000d dddd 0100
  (iii) 1001 000d dddd 0101
- Words: 1
- Cycles: 3
- Comments: Z contains the byte address while the flash memory uses word addressing. Therefore, the word address must be converted into byte address before having access to data on flash memory.

12

12

Load Program Memory (2/3)

- Example

```
    ldi zh, high(Table_1<<1)        ; Initialize Z pointer
    ldi zl, low(Table_1<<1)
    lpm r16,  z+                    ; r16=0x76
    lpm r17, z                      ; r17=0x58
     ...
Table_1: .dw 0x5876
     …
```

- Comments: Table_1<<1 converts word address into byte address

13

13

Load Program Memory (3/3)

- Example

```
    ldi zh, high(Table_1<<1)        ; Initialize Z pointer
    ldi zl, low(Table_1<<1)
    lpm r16,  z+                    ; r16=0x76
    lpm r17, z                      ; r17=0x58
     ...
Table_1: .dw 0x5876
     …
```

Comments: Table_1<<1, i.e. Table*2, converts the word address of 0x5876 into the byte address.

14

14

Load an I/O Location to Register

- Syntax: in Rd, A
- Operands: Rd∈{r0, r1, …, r31} and 0≤A≤63
- Operation: Rd←I/O (A)
  Loads one byte from the location A in the I/O Space (Ports, Timers, Configuration registers etc.) into register Rd in the register file.
- Flag affected: None
- Encoding: 1011 0AAd dddd AAAA
- Words: 1
- Cycles: 1
- Example:

```
        in r25, $16      ; Read Port B
        cpi r25, 4       ; Compare read value to constant
        breq exit        ; Branch if r25=4
        ...
exit:   nop              ; Branch destination (do nothing)
```

15

15

Store Register to an I/O Location

- Syntax: out A, Rr
- Operands: Rr∈{r0, r1, …, r31} and 0≤A≤63
- Operation: I/O (A)←Rr
  Store the byte in register Rr to the I/O location (register).
- Flag affected: None
- Encoding: 1011 1AAr rrrr AAAA
- Words: 1
- Cycles: 1
- Example:

```
clr r16          ; Clear r16
ser r17          ; Set r17 to $ff
out $18, r16     ; Write zeros to Port B
nop              ; Wait (do nothing)
out $18, r17     ; Write ones to Port B
```

16

16

Push Register on Stack

- Syntax: push Rr
- Operands: Rr∈{r0, r1, …, r31}
- Operation: (SP) ← Rr
  SP ← SP –1
- Flag affected: None
- Encoding: 1001 001d dddd 1111
- Words: 1
- Cycles: 2
- Example

```
            call routine      ; Call subroutine
            ...
routine:    push r14          ; Save r14 on the stack
            push r13          ; Save r13 on the stack
            ...
            pop r13           ; Restore r13
            pop r14           ; Restore r14
            ret               ; Return from subroutine
```

17

17

Pop Register from Stack

- Syntax: pop Rr
- Operands: Rr∈{r0, r1, …, r31}
- Operation: SP ← SP +1
  Rr ← (SP)
- Flag affected: None
- Encoding: 1000 000d dddd 1111
- Words: 1
- Cycles: 2
- Example

```
            call routine      ; Call subroutine
            ...
routine:    push r14          ; Save r14 on the stack
            push r13          ; Save r13 on the stack
            ...
            pop r13           ; Restore r13
            pop r14           ; Restore r14
            ret               ; Return from subroutine
```

18

18

AVR Assembly Programming: Example 4 (1/3)

The following AVR assembly program converts 5 lower-case letters in a string which stored in the program memory (FLASH memory) into the upper-case letters and stores the resulting string into the data memory (SRAM).

```
.include "m2560def.inc"
.equ size =5            ; Define size to be 5
.def counter =r17       ; Define counter to be  r17
.dseg                   ; Define a data segment
.org 0x100              ; Set the starting address of data segment to 0x100
Cap_string: .byte 5     ; Allocate 5 bytes of data memory (SRAM) to store the string of
                        ; upper-case letters.
.cseg                   ; Define a code segment
Low_string: .db "hello"  ; Define the string "hello" which is stored in the program
                         ; (Flash) memory.
```

19

19

AVR Assembly Programming: Example 4 (2/3)

```
ldi zl, low(Low_string<<1)     ; Get the low byte of the address of "h"
ldi zh, high(Low_string<<1)    ; Get the high byte of the address of "h"
ldi yl, low(Cap_string)
ldi yh, high(Cap_string)

clr counter                    ; counter=0
```

20

20

AVR Assembly Programming: Example 4 (3/3)

```
main:
        lpm  r20, z+     ; Load a letter from flash memory
        subi r20, 32     ; Convert it to the capital letter
        st   y+,r20      ; Store the capital letter in SRAM
        inc  counter
        cpi  counter, size
        brlt main
loop: nop
        rjmp loop
```

21

21

AVR Assembly Programming: Example 5 (1/7)

Convert the following C program into an equivalent AVR Assembly program:

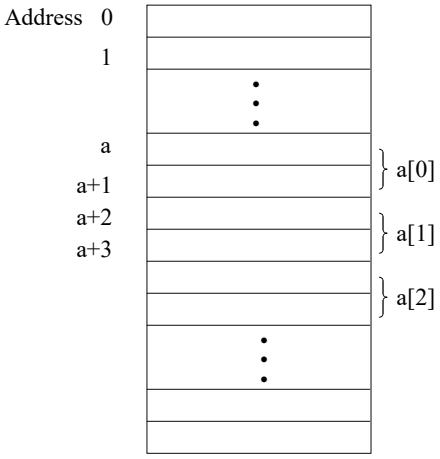
```
int a[100], max;
int i;
int main()
  {
    for (i=0; i<100; i++)
        a[i]=100*i;
    max=a[0];
    for (i=1; i<100; i++)
        if (a[i]> max) max=a[i];
    return 0;
  }
```

22

22

AVR Assembly Programming: Example 5 (2/7)

Data are stored in the SRAM. A one-dimensional array is stored consecutively in the SRAM, i.e. the element i+1 immediately follows the element i. The following figure shows how the array a is stored in the SRAM:

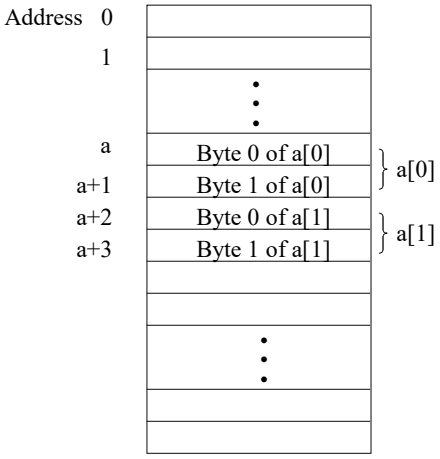


23

23

AVR Assembly Programming: Example 5 (3/7)

When a variable is more than one byte long, the programmer needs to use either big-endian order or little endian order to store its value. In this example, we use little endian order, i.e. the least significant byte is stored at the lowest address.



24

24

AVR Assembly Programming: Example 5 (4/7)

```
int a[100], max;
int i;
int main()
  {
```

```
.include "m2560def.inc"
.def  temp=r21
.def  i=r16              ; r16 stores i
.def  ai_high=r20        ; ai_high:ai_low temporarily
.def  ai_low=r19         ; store a[i]
.def  max_high=r18       ; max_high:max_low
def   max_low=r17        ; temporarily store max
.dseg.
a: .byte 200    ; Reserve 200  bytes for the array a
max: .byte 2    ; Reserve 2 byte for max
.cseg
ldi xl, low(a)     ; Let x pointer point
ldi xh, high(a)    ; the start of the array a
```

25

25

AVR Assembly Programming: Example 5 (5/7)

```
for (i=0; i<100; i++)
        a[i]=100*i;




max=a[0];
```

```
clr i          ; clear i
ldi temp, 100
forloop1: mul i, temp     ; Compute 100*i
st  x+, r0                ; a[i]=100*i
st  x+, r1
inc i
cpi i, 100
brlt forloop1
ldi xl, low(a)       ; Let x pointer point
ldi xh, high(a)      ; the start of the array a
ld max_low, x+       ; max=a[0]
ld max_high, x+
```

26

26

AVR Assembly Programming: Example 5 (6/7)

```
for (i=1; i<100; i++)
    if (a[i]> max)


          max=a[i];
```

```
ldi i, 1
forloop2: ld ai_low, x+     ; load a[i] into ai_low
ld ai_high, x+              ; and ai_high
cp max_low, ai_low          ; Compare max with a[i]
cpc max_high, ai_high
brlt lessthan
rjmp otherwise
lessthan: mov max_low, ai_low
mov max_high, ai_high
```

27

27

AVR Assembly Programming: Example 5 (7/7)

```
return 0;
    }
```

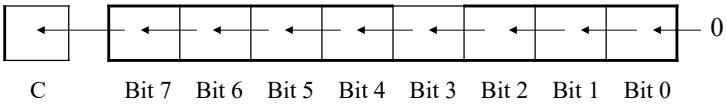
```
otherwise: inc i
cpi i, 100
brlt forloop2
ldi yl, low(max)        ; Let y pointer point to max
ldi yh, high(max)
st y+, max_low          ; Store max into SRAM
st y, max_high
Loopforever: rjmp loopforever
```

28

28

Logical Shift Left (1/2)

- Syntax: lsl Rd
- Operands: Rd ∈{r0, r1, …, r31}
- Operation: C←Rd7, Rd7← Rd6, Rd6← Rd5, …, Rd1← Rd0, Rd0←0



- Flags affected: H, S, V, N, Z, C

C ←Rd7
Set if, before the shift, the MSB of Rd was set; cleared otherwise.
N ← R7
Set if MSB of the result is set; cleared otherwise.
V←N ⊕ C
S ←N ⊕ V For signed tests.

29

29

Logical Shift Left (2/2)

- Encoding: 0000 11dd dddd dddd
- Words: 1
- Cycles: 1

- Example:
```
add r0, r4    ; Add r4 to r0
lsl r0        ; Multiply r0 by 2
```

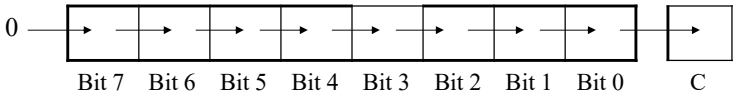
- Comments: This operation effectively multiplies a one-byte signed or unsigned integer by two.

30

30

Logical Shift Right

- Syntax: lsr Rd
- Operands: Rd ∈ {r0, r1, …, r31}
- Operation: C←Rd0, Rd0← Rd1, Rd1← Rd2, …, Rd6← Rd7, Rd7←0



- Flags affected: H, S, V, N, Z, C
- Encoding: 1001 010d dddd 0110
- Words: 1
- Cycles: 1
- Example:
```
add r0, r4    ; Add r4 to r0
lsr r0        ; Divide r0 by 2
```

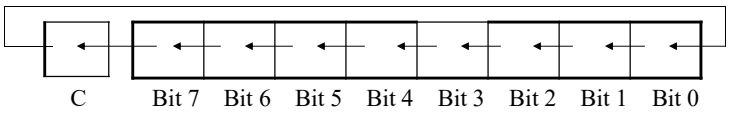
- Comments: This instruction effectively divides an unsigned one-byte integer by two. C stores the remainder.

31

31

Rotate Left Through Carry

- Syntax: rol Rd
- Operands: Rd ∈ {r0, r1, …, r31}
- Operation: temp ← C, C←Rd7, Rd7 ← Rd6, Rd6 ← Rd5, …, Rd1← Rd0, Rd0 ← temp



32

32

Rotate Left Through Carry (1/2)

- Flag affected: H, S, V, N, Z, C

C ←Rd7
Set if, before the shift, the MSB of Rd was set; cleared otherwise.
N ← R7
Set if MSB of the result is set; cleared otherwise.
V←N ⊕ C
S ←N ⊕ V For signed tests.

- Encoding: 0001 11dd dddd dddd
- Words: 1
- Cycles: 1

33

33

Rotate Left Through Carry (2/2)

- Example: Assume a 32-bit signed or unsigned integer x is stored in registers r13: r12:r11:r10. The following code computes 2*x.

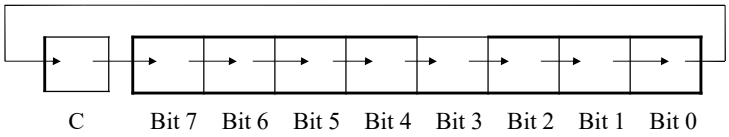
```
lsl r10     ; Shift byte 0 (least significant byte) left
rol r11     ; Shift byte 1 left through carry
rol r12     ; Shift Byte 2 left through carry
rol r13     ; Shift Byte 3 (most significant byte) left through carry
```

34

34

Rotate Right Through Carry (1/2)

- Syntax: ror Rd
- Operands: Rd ∈ {r0, r1, …, r31}
- Operation: temp ← Rd0, Rd0←Rd1, Rd1 ← Rd2, … ,
  Rd6← Rd7, Rd7 ← C, C ← temp.



- Flags affected: H, S, V, N, Z, C
- Encoding: 1001 010d dddd 0111
- Words: 1
- Cycles: 1

35

35

Rotate Right Through Carry (2/2)

- Example: Assume a 32-bit signed number x is stored in registers r13:r12:r11:r10. The following code computes x/2.

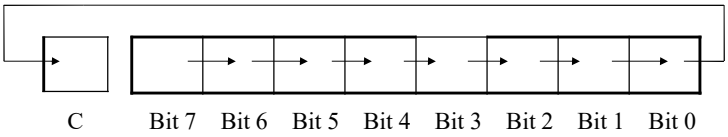
```
asr r13     ; Shift byte 3 (most significant byte) right
ror r12     ; Shift byte 2 right through carry
ror r11     ; Shift Byte 1 right through carry
ror r10     ; Shift Byte 0 (least significant byte) right through carry
```

36

36

Arithmetic Shift Right (1/2)

- Syntax: asr Rd
- Operands: Rd ∈{r0, r1, …, r31}
- Operation: C← Rd0, Rd0←Rd1, Rd1 ← Rd2, … , Rd6← Rd7



- Flag affected: H, S, V, N, Z, C
- Encoding: 1001 010d dddd 0101
- Words: 1
- Cycles: 1

37

37

Arithmetic Shift Right (2/2)

- Example

```
ldi r10, 10        ; r10=10
ldi r11, -20       ; r11=-20
add r10,  r20      ; r10=-20+10
asr r10            ; r10=(-20+10)/2
```

- Comments: This instruction effectively divides a signed value by two. C stores the remainder.

38

38

Bit Set in Status Register (1/2)

- Syntax: bset s
- Operation: Bit s of SREG (Status Register)←1
- Operands: $0 \le s \le 7$
- Flags affected

I: 1 if s = 7; Unchanged otherwise.

T: 1 if s = 6; Unchanged otherwise.

H: 1 if s = 5; Unchanged otherwise.

S: 1 if s = 4; Unchanged otherwise.

V: 1 if s = 3; Unchanged otherwise.

N: 1 if s = 2; Unchanged otherwise.

Z: 1 if s = 1; Unchanged otherwise.

C: 1 if s = 0; Unchanged otherwie.

Bit Set in Status Register (2/2)

- Encoding: 1001 0100 0sss 1000
- Words: 1
- Cycles: 1
- Example

```
bset 0          ; Set C
bset 3          ; Set V
bset 7          ; Enable interrupt
```

Bit Clear in Status Register (1/2)

- Syntax: bclr s
- Operation: Bit s of SREG (Status Register)←0
- Operands: $0 \le s \le 7$
- Flags affected

I: 0 if s = 7; Unchanged otherwise.

T: 0 if s = 6; Unchanged otherwise.

H: 0 if s = 5; Unchanged otherwise.

S: 0 if s = 4; Unchanged otherwise.

V: 0 if s = 3; Unchanged otherwise.

N: 0 if s = 2; Unchanged otherwise.

Z: 0 if s = 1; Unchanged otherwise.

C: 0 if s = 0; Unchanged otherwie.

41

41

Bit Clear in Status Register (2/2)

- Encoding: 1001 0100 1sss 1000
- Words: 1
- Cycles: 1
- Example

```
bclr 0          ; Clear C
bclr 3          ; Clear V
bclr 7          ; Disable interrupt
```

42

42

Set Bit in I/O Register

- Syntax: sbi A, b
- Operation: Bit b of the I/O register with address A←1
- Operands: 0 ≤ A ≤31 and 0 ≤ b ≤7
- Flags affected: None
- Encoding: 1001 1010 AAAA Abbb
- Words: 1
- Cycles: 2
- Example

```
out $1E, r0    ; Write EEPROM address
sbi  $1C, 0    ; Set read bit in EECR
in   r1, $1D   ; Read EEPROM data
```

43

43

Clear Bit in I/O Register

- Syntax: cbi A, b
- Operation: Bit s of the I/O register with address A←0
- Operands: 0 ≤ A ≤31 and 0 ≤ b ≤7
- Flags affected: None
- Encoding: 1001 1000 AAAA Abbb
- Words: 1
- Cycles: 2
- Example

```
cbi $12, 7      ; Clear bit 7 in Port D
```

44

44

Set Flags (1/2)

- Syntax: sex
  where x ∈{I, T, H, S, V, N, Z, C}
- Operation: Flag x←1
- Operands: None
- Flags affected: Flag x←1
- Encoding: Depends on x.
  Refer to the AVR Instruction Set for details of encoding.
- Words: 1
- Cycles: 1

45

45

Set Flags (2/2)

- Example

```
sec             ; Set carry flag
adc r0, r1      ; r0=r0+r1+1
sec
sbc r0, r1      ; r0=r0–r1–1
sen             ; Set negative flag
sei             ; Enable interrupt
sev             ; Set overflow flag
sez             ; Set zero flag
ses             ; Set sign flag
```

46

46

Clear Flags

- Syntax: clx

  where x ∈{I, T, H, S, V, N, Z, C}
- Operation: Flag x←0
- Operands: None
- Flags affected: Flag x←0
- Encoding: Depends on x.

  Refer to the AVR Instruction Set for details of encoding.
- Words: 1
- Cycles: 1

47

47

Clear Flags

- Example

```
clc     ; Clear carry flag
cln     ; Clear negative flag
cli     ; Disable interrupt
clv     ; Clear overflow flag
clz     ; Clear zero flag
cls     ; Clear sign flag
```

48

48

No Operation

- Syntax: nop
- Operation: No
- Operands: None
- Flags affected: None
- Encoding: 0000 0000 0000 0000
- Words: 1
- Cycles: 1
- Example

```
clr   r16          ; Clear r16
ser  r17           ; r17=0xff
out $18, r16       ; Write zeros to Port B
nop                ; Wait (do nothing)
out $18, r17       ; Write ones to Port B
```

49

49

Sleep

- Syntax: sleep
- Operation: Sets the circuit in sleep mode defined by the MCU control register. When an interrupt wakes the MCU from the sleep mode, the instructions following the sleep instruction will be executed.
- Operands: None
- Flags affected: None
- Encoding: 1001 0101 1000 1000
- Words: 1
- Cycles: 1
- Example

```
mov r0, r11          ; Copy r11 to r0
ldi r16, (1<<SE)     ; Enable sleep mode (SE=5)
out MCUCR, r16
sleep                ; Put MCU in sleep mode
```

50

50

Watchdog Reset

• Syntax: wdr

• Operation: Resets the Watchdog Timer. This instruction must be executed within a limited time given by the WD prescaler. See the Watchdog Timer hardware specification.

• Operands: None

• Flags affected: None

• Encoding: 1001 0101 1010 1000

• Words: 1

• Cycles: 1

• Example

```
wdr          ; Reset watchdog timer
```

51

51

Break

• Syntax: break

• Operation: The break instruction is used by the On-Chip Debug system, and is normally not used in the application software. When the BREAK instruction is executed, the AVR CPU is set in the Stopped Mode. This gives the On-Chip Debugger access to internal resources.

If any lock bits are set, or either the JTAGEN or OCDEN fuses are unprogrammed, the CPU will treat the break instruction as a nop and will not enter the Stopped Mode.

• Operands: None

• Flags affected: None

• Encoding: 1001 0101 1001 1000

• Words: 1

• Cycles: 1

• Example

```
break        ; stop here
```

52

52

Assembly Programming: Example 6 (1/8)

Write an AVR assembly program to implement the following C program where all elements of the array a, b and c are two bytes long.

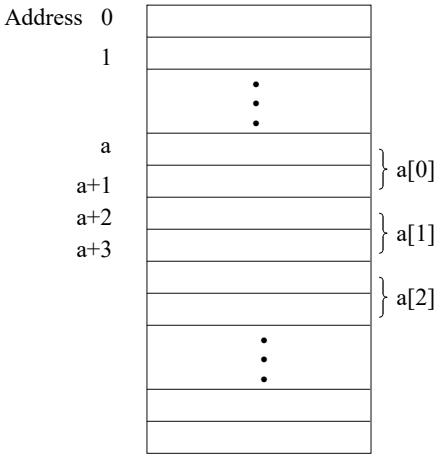
```
int a[10], b[10], c[10];
void main()
{ int i;
  for (i=0; i++; i<10)
    { a[i]= –4*i*i;
      b[i]= –5*i*i;
    }
  for (i=0; i++; i<9)
    c[i]=b[i]+a[i];
}
```

53

53

Assembly Programming: Example 6 (2/8)

Data are stored in the SRAM. A one-dimensional array is stored consecutively in the SRAM, i.e. the element i+1 immediately follows the element i. The following figure shows how the array a is stored in the SRAM:

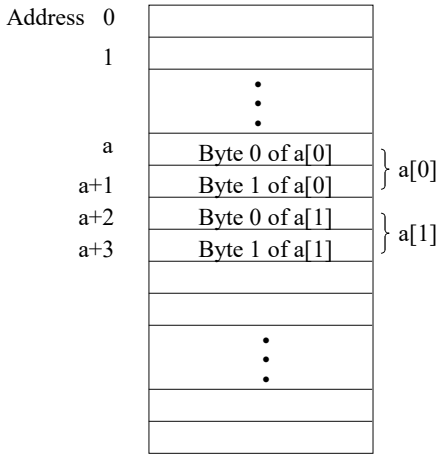


54

54

Assembly Programming: Example 6 (3/8)

When a variable is more than one byte long, the programmer needs to use either big-endian order or little endian order to store its value. In this example, we use little endian order, i.e. the least significant byte is stored at the lowest address.



55

55

Assembly Programming: Example 6 (4/8)

```
.include "m2560def.inc
.def  i=r20
.dseg
a: .byte 20            ; Allocate 20 bytes to the array a
b: .byte 20
c: .byte 20
.cseg
ldi xl, low(a)      ; load byte 0 of the starting address of the array a
ldi xh, high(a)     ; load byte 1
ldi yl, low(b)
ldi yh, high(b)
clr i               ; i is the loop counter
```

56

56

Assembly Programming: Example 6 (5/8)

```
loop1:  mov r16, i
        mov r17, i
        mul r17, r16
        movw r17:r16, r1:r0      ; r17:r16=i*i
        clr r19
        clr r18
        sub r18, r16
        sbc r19, r17             ; r19:r18= -i*i
        movw r17:r16, r19:r18
        lsl r16
        rol r17                  ; r17:r16=-2*i*i
```

57

57

Assembly Programming: Example 6 (6/8)

```
        lsl r16
        rol r17                  ; r17:r16= -4*i*i
        st x+, r16
        st x+, r17               ; a[i]= -4*i*i
        add r18, r16
        adc r19, r17             ; r19:r18= -5*i*i
        st y+, r18
        st y+, r19               ; b[i]= -5*i*i
        inc i
        cpi i, 9
        brlo loop1
```

58

58

Assembly Programming: Example 6 (7/8)

```
        ldi xl,low(a)
        ldi xh, high(a)
        ldi yl, low(b)
        ldi yh, high(b)
        ldi zl, low(c)
        ldi zh, high(c)
        clr i
loop2:
        ld r17, x+     ; load byte 0 of a[i]
        ld r18, x+     ; load byte 1 0f a[i]
        ld r15, y+     ; load byte 0 of b[i]
        ld r16, y+     ;  load byte 0 of b[i]
```

59

59

Assembly Programming: Example 6 (8/8)

```
        add r15, r17
        adc r16, r18          ; r16:r15=a[i]+b[i]
        st z+, r15            ; store byte 0 of c[i]
        st z+, r16            ; store byte 1 of c[i]
        inc i
        cpi i, 9
        brlo loop2
loop:  rjmp loop
```

60

60

Reading Material

1. AVR Instruction Set (http://www.cse.unsw.edu.au/~cs2121/AVR/AVR-Instruction-Set.pdf)
2. AVR Assembler User Guide (http://www.cse.unsw.edu.au/~cs2121/AVR/AVR-Assembler-Guide.pdf)

61