CprE 288 – Introduction to Embedded Systems
ATmega128 Assembly Programming: Moving Data & Control Flow

Instructors:
Dr. Phillip Jones (Sections F, G, J)
Dr. Zhao Zhang (Sections A, B, C, D, E)

Major Classes of Assembly Instructions

- Data Movement
  - Move data between registers
  - Move data in & out of SRAM
  - Different addressing modes
- Logic & Arithmetic
  - Addition, subtraction, etc.
  - AND, OR, bit shift, etc.
- Control Flow
  - Control which sections of code should be executed (e.g. In C "IF", "CASE", "WHILE", etc.
  - Typically the result of Logic & Arithmetic instructions help decided what path to take through the code.

Major Classes of Assembly Instructions

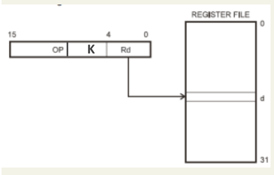
- Data Movement
  - Move data between registers
  - Move data in & out of SRAM
  - Different addressing modes
- Logic & Arithmetic
  - Addition, subtraction, etc.
  - AND, OR, bit shift, etc.
- Control Flow
  - Control which sections of code should be executed (e.g. In C "IF", "CASE", "WHILE", etc.
  - Typically the result of Logic & Arithmetic instructions help decided what path to take through the code.

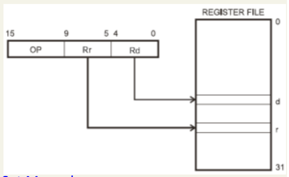
Instructions to move data: Summary

- LDI Rd, K Load Immediate *Rd ← K* 1 clk
- MOV Rd, Rr Move Between Registers *Rd ← Rr* 1 clk
- LDS Rd, (k) Load Direct *Rd ← (k)* 2 clks
  - *Note: There is a ST version for each LD (except for LDI)*
- LD Rd, Y Load Indirect *Rd ← (X)* 2 clks
- LD Rd, Y+ Load Indirect & Post-Inc. *Rd←(X), X← X + 1* 2clks
- LDD Rd, Y+q Load Indirect + offset. *Rd ←(X+q)* 2 clks

LDI & MOV

- LDI Rd, K Load Immediate *Rd ← K* 1 clk
- MOV Rd, Rr Move Between Registers *Rd ← Rr* 1 clk
- Only need one clock cycle to execute
  - All parameters needed for execution available to ALU





- Reference: (doc0856) AVR Instruction Set Manual
  - Pages 3-7

Load Immediate

```
char a;
…
a = 0x10;

ldi     r24, 0x10     ; Load imm 10
sts     a, r24        ; Store to a
```

Load Immediate

LDI: Load an 8-bit constant (limited to R16-R31)

Syntax: **LDI Rd, K**
Operands: 16 ≤ d ≤ 31, 0 ≤ K ≤ 255
Operations: Rd ← K, PC ← PC+1
Binary Format

1110	KKKK	dddd	KKKK

Cycles: 1
See 8-bit AVR Inst. Set Page 89
Question: Why limited to R16-R31?

Copy Register

MOV: Copy one register to another

Syntax: `MOV Rd, Rr`
Operands: 0 ≤ d ≤ 31, 0 ≤ r ≤ 31
Operations: Rd ← Rr, PC ← PC+1
Example:
mov r16,r0 ; Copy r0 to r16

Copy Register Word

MOVW: Copy one register to another

Syntax (AVR): `MOVW Rd+1:Rd, Rr+1:Rr`
Syntax (GCC): `MOVW Rd, Rr`

Operands: d=0,2,…,30, r=0,2,…,30
Operations: Rd+1:Rd ← Rr+1:Rr, PC ← PC+1
Example:
(AVR) movw r17:16, r1:r0
(GCC) movw r16, r0

Copy Register & Copy Register Word

Make R2 = 0x10
– Recall: Cannot use LDI on R2

```
ldi   r24, 0x10     ; r24 = 0x10
mov   r2,  r24      ; r2 = r24
```

Make R5:R4 = 0x3020 using three instructions

```
ldi   r24, 0x20     ; r24 = 0x20
ldi   r25, 0x30     ; r25 = 0x30
movw  r4,  r24      ; r5:r4=r25:r24
```

Exercise

```
int a;
…
a = 0x2030;
```

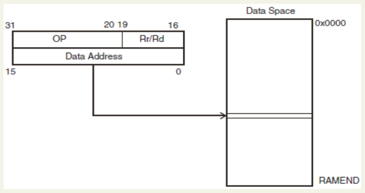
Load and Store

```
int a;
…
a = 0x2030;

ldi r24, 0x30      ; r24 = 0x30
sts a, r24         ; Store to lower half
ldi r24, 0x20      ; r24 = 0x20
sts a+1, r24       ; Store to higher half
```

LDS (Load Direct from Data Space)

- LDS Rd, (k)   Load Direct *Rd ← (k)*   2 clks
- Note:



Load and Store

```
char a, b;
…
a = b;

lds r24, b          ; Load from b
sts a, r24          ; Store to a
```

14

Load Direct

LDS: Load direct from storage space

Syntax: LDS Rd, k
Operands: 0 ≤ d ≤ 31, 0 ≤ k ≤ 65,535
Operations: Rd ← (k), PC ← PC+2
Binary
  Format

1001	000d	dddd	0000
kkkk	kkkk	kkkk	kkkk

Cycles: 2
See 8-bit AVR Inst. Set Page 90

Store Direct

STS: Store direct to storage space

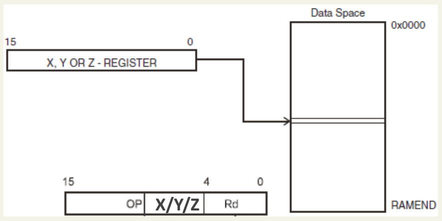
Syntax: STS k, Rr
Operands: 0 ≤ r ≤ 31, 0 ≤ k ≤ 65,535
Operations: (k) ← Rr, PC ← PC+2
Binary
  Format

1001	001d	dddd	0000
kkkk	kkkk	kkkk	kkkk

Cycles: 2
See 8-bit AVR Inst. Set Page 113

LD (Load Indirect from Data Space)

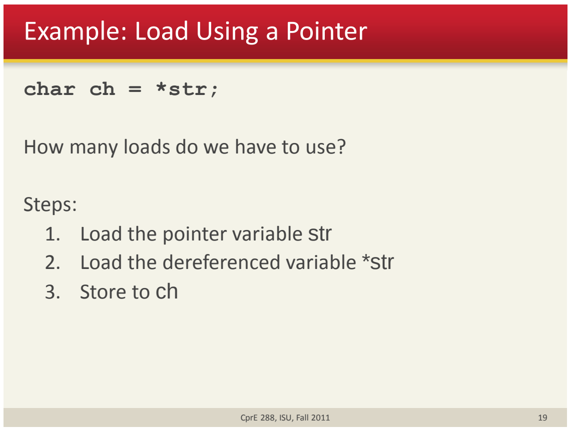
- LD Rd, X   Load Indirect *Rd ← (X)*



X, Y, Z Registers

Three **indirect address (pointer) registers**: X, Y, and Z
  X ⇔ R27:R26
  Y ⇔ R29:R28
  Z ⇔ R31:R30

Use the GPR names to manipulate the pointers
Use the X, Y, Z names to dereference



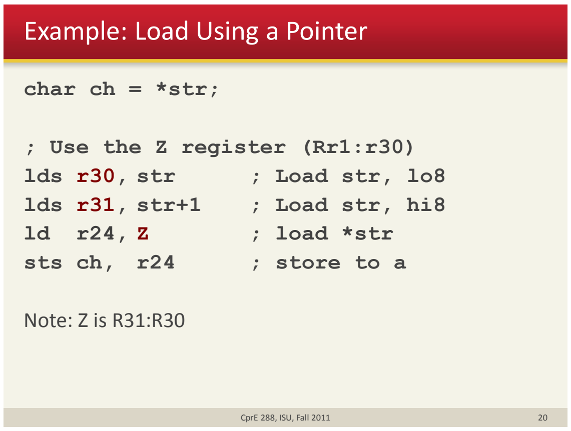
Example: Load Using a Pointer

```
char ch = *str;
```

How many loads do we have to use?

Steps:

1. Load the pointer variable str
2. Load the dereferenced variable *str
3. Store to ch

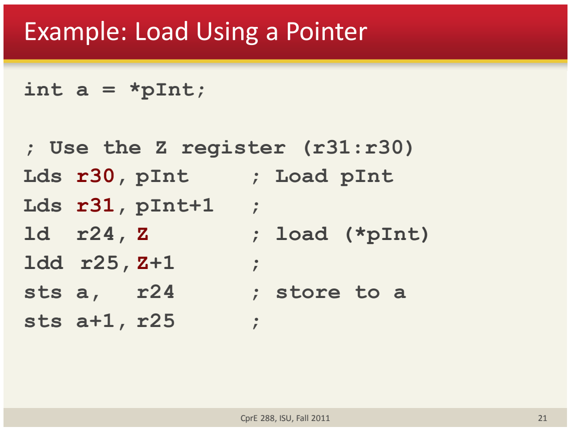


Example: Load Using a Pointer

```
char ch = *str;

; Use the Z register (Rr1:r30)
lds r30, str       ; Load str, lo8
lds r31, str+1     ; Load str, hi8
ld  r24, Z         ; load *str
sts ch,  r24       ; store to a
```

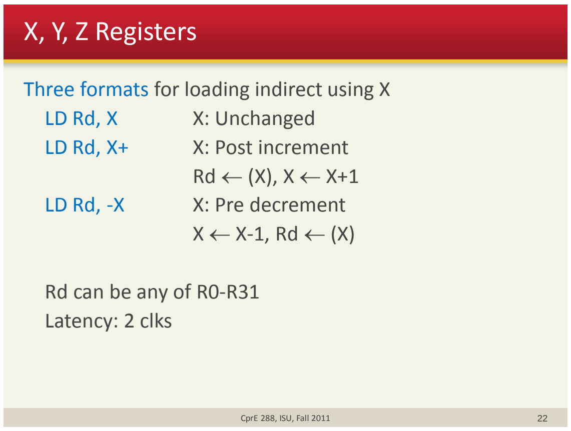
Note: Z is R31:R30



Example: Load Using a Pointer

```
int a = *pInt;

; Use the Z register (r31:r30)
Lds r30, pInt      ; Load pInt
Lds r31, pInt+1    ;
ld  r24, Z         ; load (*pInt)
ldd r25, Z+1       ;
sts a,   r24       ; store to a
sts a+1, r25       ;
```



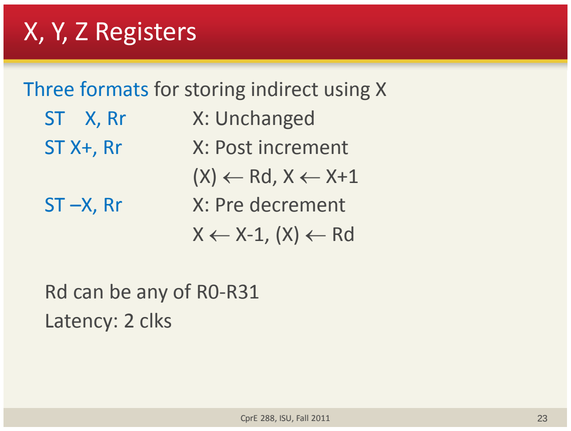
X, Y, Z Registers

Three formats for loading indirect using X

LD Rd, X	X: Unchanged
LD Rd, X+	X: Post increment Rd ← (X), X ← X+1
LD Rd, -X	X: Pre decrement X ← X-1, Rd ← (X)

Rd can be any of R0-R31

Latency: 2 clks



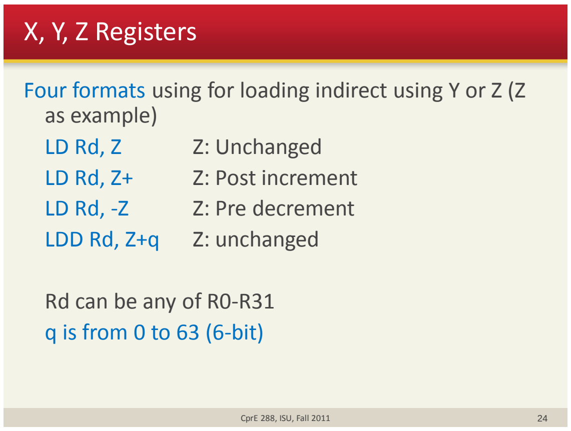
X, Y, Z Registers

Three formats for storing indirect using X

ST X, Rr	X: Unchanged
ST X+, Rr	X: Post increment (X) ← Rd, X ← X+1
ST –X, Rr	X: Pre decrement X ← X-1, (X) ← Rd

Rd can be any of R0-R31

Latency: 2 clks



X, Y, Z Registers

Four formats using for loading indirect using Y or Z (Z as example)

LD Rd, Z	Z: Unchanged
LD Rd, Z+	Z: Post increment
LD Rd, -Z	Z: Pre decrement
LDD Rd, Z+q	Z: unchanged

Rd can be any of R0-R31

q is from 0 to 63 (6-bit)

X, Y, Z Registers

Four formats for storing indirect using Y and Z (Z as example)

ST Z, Rr	Z: Unchanged
ST Z+, Rr	Z: Post increment
ST -Z, Rr	Z: Pre decrement
STD Z+q, Rr	Z: unchanged

Rd can be any of R0-R31

q is from 0 to 63 (6-bit)

Example of Encoding

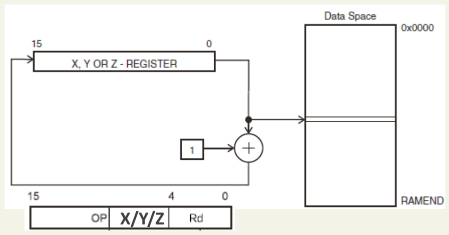
Syntax (using Z): LDD Rd, Z+q
Operands: 0 ≤ d ≤ 31, 0 ≤ q ≤ 63
Operations: Rd ← (Z+q), PC ← PC+1
Binary Format

10q0	qq0d	dddd	0qqq

Cycle: 2
See 8-bit AVR Inst. Set Page 88
Note: Unique coding for X, Y, and Z
Where is the index of Z (R31:R30)?

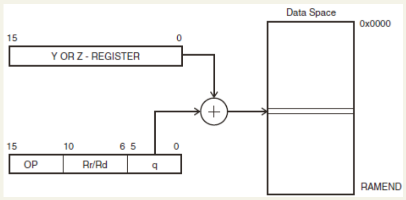
LD X+ (LoaD indirect & post increment)

- LD Rd, X+ Load Indirect & Post-Inc. *Rd←(X), X← X + 1*
  2 clks



LDD Y+q (LoaD indirect with Displacement)

- LDD Rd, Y+q Load Indirect + offset. *Rd ←(Y+q)* 2 clks



- What is the significance of q being 6-bits in size?

Array Access

```
extern int A[], B[];
A[0] = B[0];
```

First initialize X and Z registers: RegX=A, RegZ=B

```
  ldi r26,    lo8(A)   ; RegX = A
  ldi r27,    hi8(A)   ;
  ldi r30,    lo8(B)   ; RegZ = B
  ldi r31,    hi8(B)   ;
```

Recall, array names are address constants
Note: lo8 and hi8 are gcc assembly macros

Array Access

Then, load B[0] and store to A[0]

```
  ld   r24,   Z+        ; r25:r24 = B[0]
  ld   r25,   Z+        ;
  st   X+,    r24       ; A[0] = r25:r24
  st   X+,    r25       ;
```

The whole array can be copied if the code continues

Major Classes of Assembly Instructions

- Data Movement
  - Move data between registers
  - Move data in & out of SRAM
  - Different addressing modes
- Logic & Arithmetic
  - Addition, subtraction, etc.
  - AND, OR, bit shift, etc.
- Control Flow
  - Control which sections of code should be executed (e.g. In C "IF", "CASE", "WHILE", etc.
  - Typically the result of Logic & Arithmetic instructions help decided what path to take through the code.

Arithmetic Instruction

Overview of arithmetic instructions
Addition: ADD, ADC, ADIW
Subtraction: SUB, SUBI, SBC, SBCI, SBIW
Logic: AND, ANDI, OR, ORI, EOR
Compliments: COM, NEG
Register Bit Manipulation: SBR, CBR
Register Manipulation: INC, DEC, TST, CLR, SER
Multiplication1: MUL, MULS, MULSU
Fractional Multiplication1: FMUL, FMULS, FMULSU
Source: AVR Studio 4 and ATmega128: A Beginner's Guide, Page 31

Arithmetic Instruction

```
int a, b;
…
a = a + b;
```

Arithmetic Instruction

```
lds   r18, a     ; load a
lds   r19, a+1   ;
lds   r24, b     ; load b
lds   r25, b+1   ;
add   r24, r18   ; add lower half
adc   r25, r19   ; add higher half
sts   a+1, r25   ; store a.byte1
sts   a, r24     ; store a.byte0
```

Add without Carry

ADD: Add two registers without carry

Syntax: ADD Rd, Rr
Operands: 0 ≤ d ≤ 31, 0 ≤ r ≤ 31
Operations: Rd ← Rd+Rr, PC ← PC+1
Binary Format

0000	11rd	dddd	rrrr

SREG

I	T	H	S	V	N	Z	C
–	–	⇔	⇔	⇔	⇔	⇔	⇔

Add with Carry

ADC: Add two registers with carry

Syntax: ADC Rd, Rr
Operands: 0 ≤ d ≤ 31, 0 ≤ r ≤ 31
Operations: Rd ← Rd+Rr+C, PC ← PC+1
Binary Format

0001	11rd	dddd	rrrr

SREG

I	T	H	S	V	N	Z	C
–	–	⇔	⇔	⇔	⇔	⇔	⇔

Arithmetic Instruction

```
int a;
…
a -= 0x4321;
```

Arithmetic Instruction

```
lds   r24, a    ; load from a
lds   r25, a+1  ;
subi  r24, 0x21 ; sub imm 0x21
sbci  r25, 0x43 ; sub imm 0x43 with
                ; carry
sts   a+1, r25  ; store a
sts   a, r24    ;
```

Subtract Immediate

SUBI: Subtract a register and a constant

Syntax: SUBI Rd, K

Operands: 16 ≤ d ≤ 31, 0 ≤ K ≤ 255

Operations: Rd ← Rd-K, PC ← PC+1

Binary Format

0101	KKKK	dddd	KKKK

SREG

I	T	H	S	V	N	Z	C
–	–	⇔	⇔	⇔	⇔	⇔	⇔

Subtract Immediate with Carry

SBCI: Add two registers with carry

Syntax: SBCI Rd, K

Operands: 16 ≤ d ≤ 31, 0 ≤ K ≤ 255

Operations: Rd ← Rd-K-C, PC ← PC+1

Binary Format

0100	KKKK	dddd	KKKK

SREG

I	T	H	S	V	N	Z	C
–	–	⇔	⇔	⇔	⇔	⇔	⇔

Arithmetic Instruction

```
char a;
…
a += 0x0F;
```

How to write the assembly code?

Challenge: There are no "ADDI" and "ADIC"!

Logical AND

AND: Logical AND of two registers

Syntax: AND Rd, Rr

SREG

I	T	H	S	V	N	Z	C
–	–	–	⇔	0	⇔	⇔	–

Example:

and r2,r3 ; Bitwise and r2 and r3, result in r2

ldi r16,1 ; Set bitmask 0000 0001 in r16

and r2,r16 ; Isolate bit 0 in r2

Logical AND with Immediate

ANDI: Logical AND of a register and a constant

Syntax: ANDI Rd, K (16≤r≤31, 0≤K≤255)

SREG

I	T	H	S	V	N	Z	C
–	–	⇔	⇔	⇔	⇔	⇔	⇔

Example:

andi r17,$0F ; Clear upper nibble of r17

andi r18,$10 ; Isolate bit 4 in r18

andi r19,$AA ; Clear odd bits of r19

Multiply Unsigned

MUL: Multiply unsigned two registers

Syntax: MUL Rd, Rr

Operation: R1:R0 ← Rd × Rr (unsigned)

SREG

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	⇔	⇔

Example:

mul r5,r4 ; Multiply unsigned r5 and r4

movw r4,r0 ; Copy result back in r5:r4

Major Classes of Assembly Instructions

- Data Movement
  - Move data between registers
  - Move data in & out of SRAM
  - Different addressing modes
- Logic & Arithmetic
  - Addition, subtraction, etc.
  - AND, OR, bit shift, etc.
- Control Flow
  - Control which sections of code should be executed (e.g. In C "IF", "CASE", "WHILE", etc.
  - Typically the result of Logic & Arithmetic instructions help decided what path to take through the code (i.e. they set flags)

Control Flow Instruction Examples

Set Flags	CPI Rd,K	Compare with Imm	Rd - K
	CP Rd,Rr	Compare	Rd - Rr
	CPC Rd,Rr	Compare with Carry	Rd - Rr – C
Branch on conditions	BREQ k	Branch if Equal	if (Z= 1) then PC ← PC + k + 1
	BRNE k	Branch if Not Equal	if (Z= 0) then PC ← PC + k + 1
	BRIE k	Branch if Interrupt Enabled	if (I = 1) then PC ← PC + k + 1
Unconditional branch	RJMP k	RJMP k	PC ← PC + k + 1
	IJMP	Indirect Jump to (Z)	PC ← Z
	RCALL k	Relative Call Subroutine	PC ← PC + k + 1
	RET	Subroutine Return	PC ← STACK
	RETI	Interrupt Return	PC ← STACK
Compare & Branch (skip next) on condition	CPSE Rd,Rr	Compare, Skip if Equal	if (Rd = Rr) PC ← PC + 2
	SBRC Rr, b	Skip if Bit in Register Cleared	if (Rr(b)=0) PC ← PC + 2
	SBRS Rr, b	Skip if Bit in Register Set	if (Rr(b)=1) PC ← PC + 2

Summary of Branch Conditions

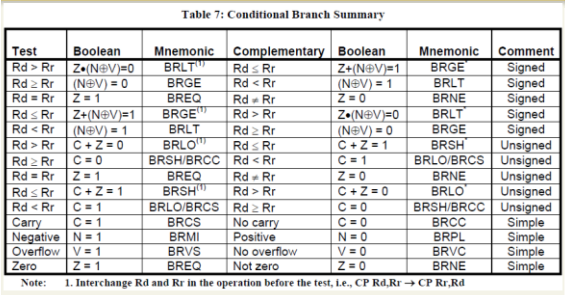


Table 7: Conditional Branch Summary

Test	Boolean	Mnemonic	Complementary	Boolean	Mnemonic	Comment
Rd > Rr	Z•(N⊕V)=0	BRLT(1)	Rd ≤ Rr	Z+(N⊕V)=1	BRGE'	Signed
Rd ≥ Rr	(N⊕V) = 0	BRGE	Rd < Rr	(N⊕V) = 1	BRLT	Signed
Rd = Rr	Z = 1	BREQ	Rd ≠ Rr	Z = 0	BRNE	Signed
Rd ≤ Rr	Z+(N⊕V)=1	BRGE(1)	Rd > Rr	Z•(N⊕V)=0	BRLT'	Signed
Rd < Rr	(N⊕V) = 1	BRLT	Rd ≥ Rr	(N⊕V) = 0	BRGE	Signed
Rd > Rr	C + Z = 0	BRLO(1)	Rd ≤ Rr	C + Z = 1	BRSH'	Unsigned
Rd ≥ Rr	C = 0	BRSH/BRCC	Rd < Rr	C = 1	BRLO/BRCS	Unsigned
Rd = Rr	Z = 1	BREQ	Rd ≠ Rr	Z = 0	BRNE	Unsigned
Rd ≤ Rr	C + Z = 1	BRSH(1)	Rd > Rr	C + Z = 0	BRLO'	Unsigned
Rd < Rr	C = 1	BRLO/BRCS	Rd ≥ Rr	C = 0	BRSH/BRCC	Unsigned
Carry	C = 1	BRCS	No carry	C = 0	BRCC	Simple
Negative	N = 1	BRMI	Positive	N = 0	BRPL	Simple
Overflow	V = 1	BRVS	No overflow	V = 0	BRVC	Simple
Zero	Z = 1	BREQ	Not zero	Z = 0	BRNE	Simple

Note: 1. Interchange Rd and Rr in the operation before the test, i.e., CP Rd,Rr → CP Rr,Rd

- Reference: (doc0856) AVR Instruction Set Manual
  - Page 10

Control Instruction Examples: IF

```
 R17  R2
if (X == Y) X = X+Y;
```

```
         CP R17, R2    ; compare X & Y
         BRNE SKIPADD  ; Z=1 iff X==Y
         ADD R17, R2   ; X = X+Y
SKIPADD: ADD R1, R2    ; do something
```

Control Instruction Examples: IF/Else

```
  R17   R2
if (X == Y) X = X+Y;
 else X = X-Y;

         CP R17, R2        ; compare X & Y
         BREQ  THENPART    ; Z=1 iff X==Y
         SUB R17, R2; X = X-Y : ELSEPART
         JMP SKIPTHENPART  ;do not do THENPART
                           ;as well
THENPART: ADD R17, R2      ; X=X+Y
SKIPTHENPART:
```

While/For LOOP

```
        R17              R2   R1
for (i=0; i < 100; i++) X = X+Y;

      CLR R17         ;i=0
LOOP: ADD R2, R1      ; X=X+Y
      INC  R17        ;i=i+1
      CPI  R17, 100   ; i-100
      BRLT  LOOP      ; branch on: N flag = 1
```

While/For LOOP: Can we do better?

```
        R17              R2   R1
for (i=0; i < 100; i++) X = X+Y;

      LDI R17, 100    ; i = 100
LOOP: ADD R2, R1      ; X=X+Y
      DEC  R17        ;i=i-1
      BRNE   LOOP     ; branch on not zero
```

Processing an Array

```
char A[100]; // Assume address is at 0x0200 ( X )
for (i=0; i < 100; i++)A[i] = A[i]+1;
       R17              R16

                         ; X is R27:R26
      LDI R26, 0x00      ;XL holds LB of address A
      LDI R27, 0x02      ;XH holds MB of address A
      LDI R17, 100       ; i = 100
LOOP: LD R16, X          ; A[i]
      INC R16            ; A[i]=A[i]+1
      ST X, R16
      ADIW R26, 1        ; next A[i]
      DEC  R17           ;i=i-1
      BRNE   LOOP        ; branch on not zero
```

Processing an Array

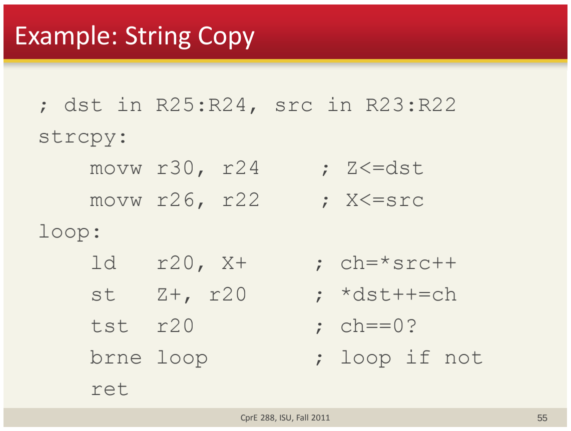
```
char A[100]; // Assume address is at 0x0200 ( X )
for (i=0; i < 100; i++)A[i] = A[i]+1;
       R17              R16

                         ; X is R27:R26
      LDI R26, 0x00      ;XL holds LB of address A
      LDI R27, 0x02      ;XH holds MB of address A
      LDI R17, 100       ; i = 100
LOOP: LD R16, X          ; A[i]
      INC R16            ; A[i]=A[i]+1
      ST X+, R16
      ~~ADIW R26, 1        ; next A[i]~~
      DEC  R17           ;i=i-1
      BRNE   LOOP        ; branch on not zero
```

Example: String Copy

```
void strcpy (char *dst, char *src)
{
    char ch;

    do {
        ch = *src++;
        *dst++ = ch;
    } while (ch);
}
```



Example: String Copy

```
; dst in R25:R24, src in R23:R22
strcpy:
    movw r30, r24     ; Z<=dst
    movw r26, r22     ; X<=src
loop:
    ld   r20, X+     ; ch=*src++
    st   Z+, r20     ; *dst++=ch
    tst  r20         ; ch==0?
    brne loop        ; loop if not
    ret
```

CprE 288, ISU, Fall 2011 55