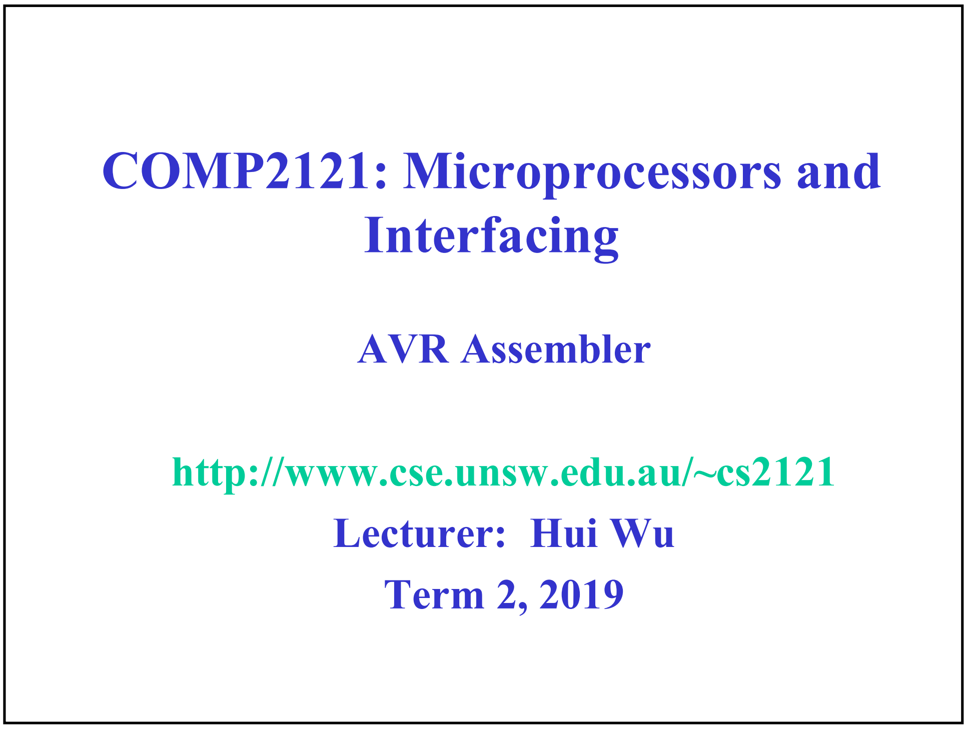


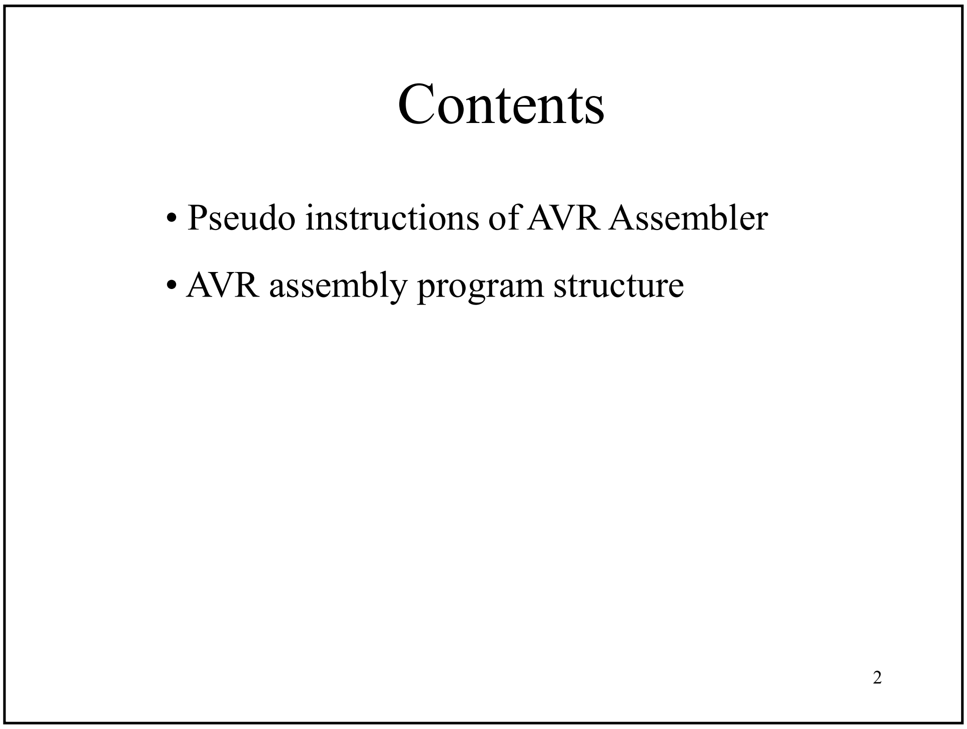
COMP2121: Microprocessors and Interfacing

AVR Assembler

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

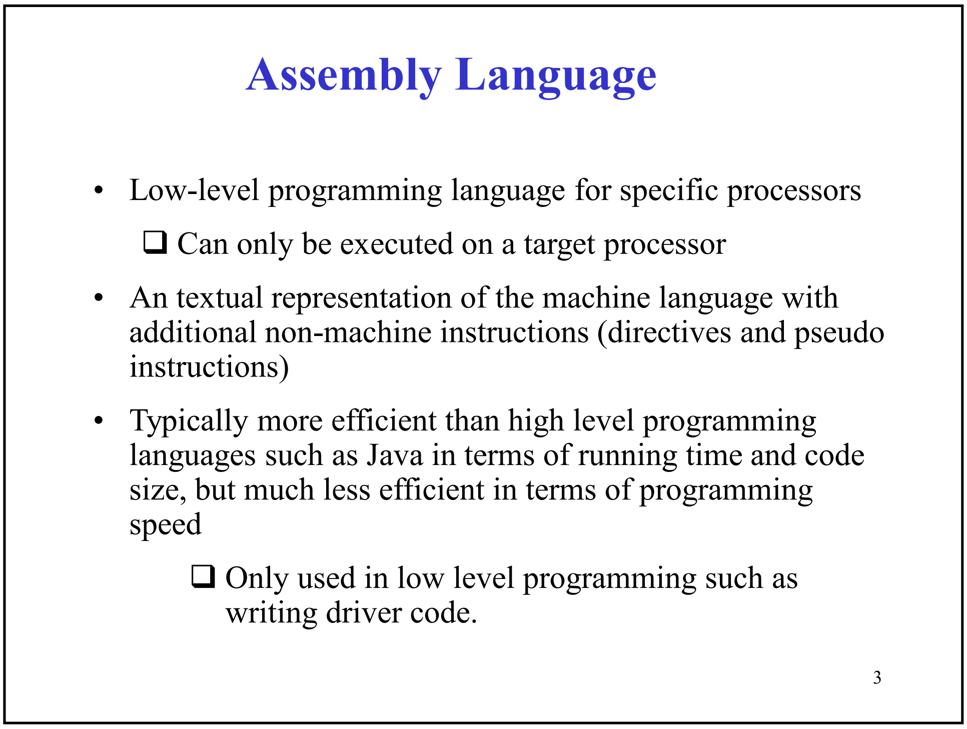
Lecturer: Hui Wu

Term 2, 2019



Contents

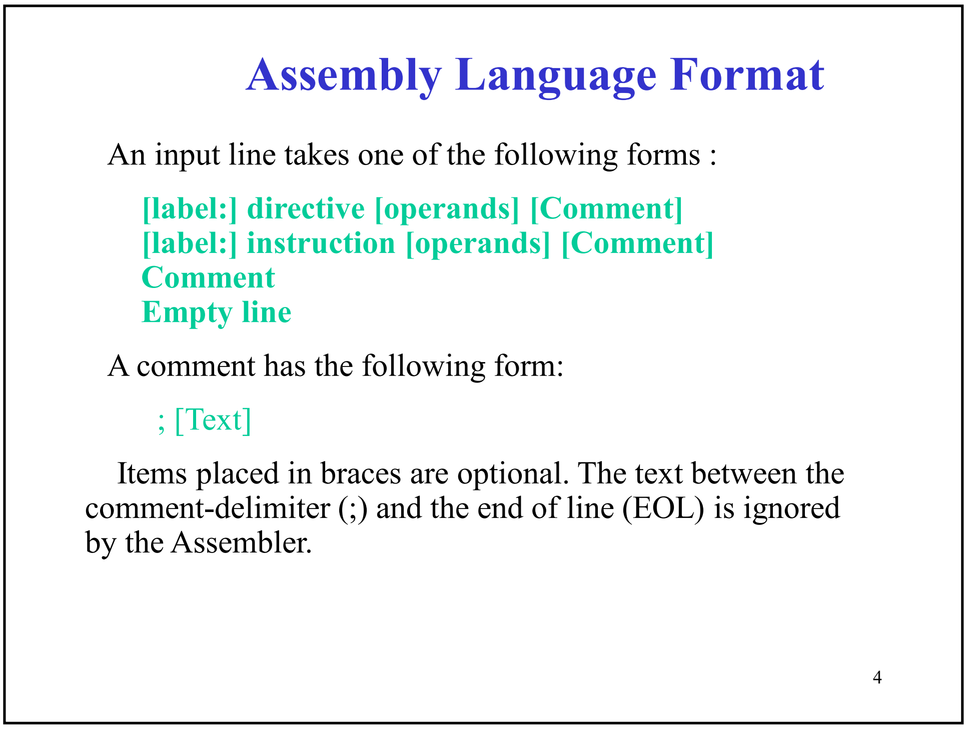
- Pseudo instructions of AVR Assembler
- AVR assembly program structure



Assembly Language

- Low-level programming language for specific processors
  - ❑ Can only be executed on a target processor
- An textual representation of the machine language with additional non-machine instructions (directives and pseudo instructions)
- Typically more efficient than high level programming languages such as Java in terms of running time and code size, but much less efficient in terms of programming speed
  - ❑ Only used in low level programming such as writing driver code.

3



Assembly Language Format

An input line takes one of the following forms :

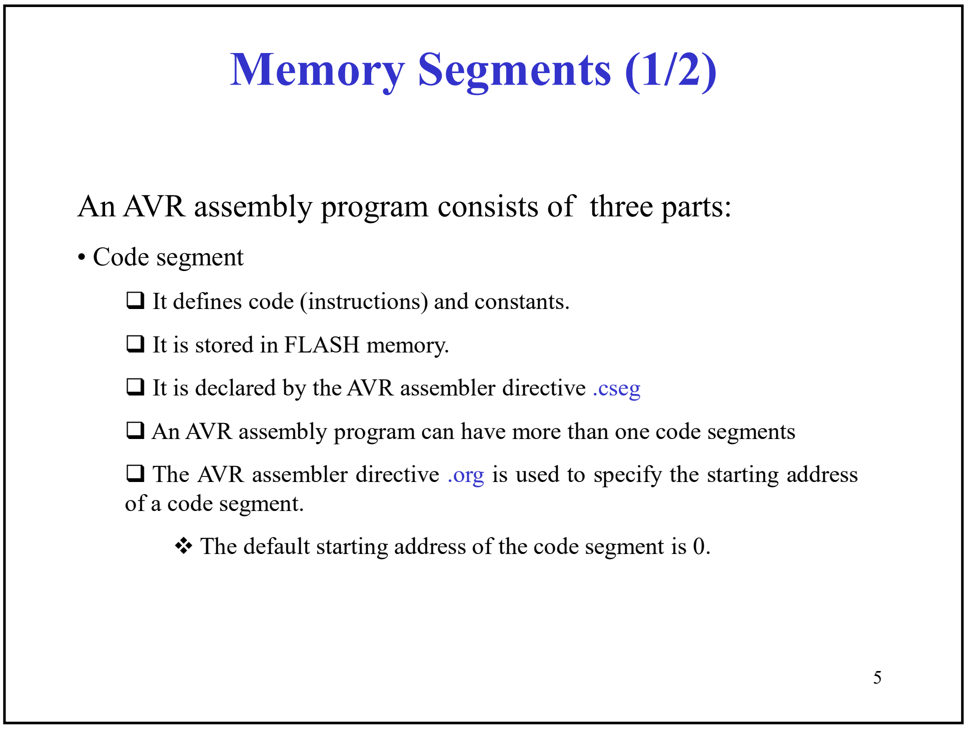
[label:] directive [operands] [Comment]
[label:] instruction [operands] [Comment]
Comment
Empty line

A comment has the following form:

; [Text]

Items placed in braces are optional. The text between the comment-delimiter (;) and the end of line (EOL) is ignored by the Assembler.

4



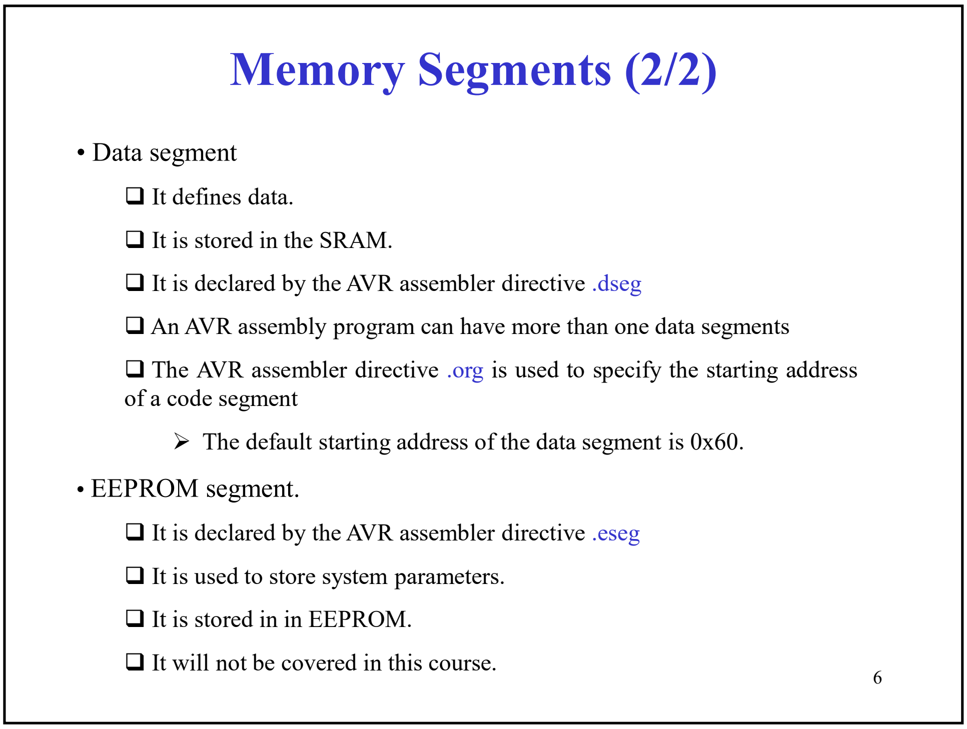
Memory Segments (1/2)

An AVR assembly program consists of three parts:

- Code segment
  - ❑ It defines code (instructions) and constants.
  - ❑ It is stored in FLASH memory.
  - ❑ It is declared by the AVR assembler directive .cseg
  - ❑ An AVR assembly program can have more than one code segments
  - ❑ The AVR assembler directive .org is used to specify the starting address of a code segment.
    - ❖ The default starting address of the code segment is 0.

5

5



Memory Segments (2/2)

- Data segment
  - ❑ It defines data.
  - ❑ It is stored in the SRAM.
  - ❑ It is declared by the AVR assembler directive .dseg
  - ❑ An AVR assembly program can have more than one data segments
  - ❑ The AVR assembler directive .org is used to specify the starting address of a code segment
    - ➢ The default starting address of the data segment is 0x60.
- EEPROM segment.
  - ❑ It is declared by the AVR assembler directive .eseg
  - ❑ It is used to store system parameters.
  - ❑ It is stored in in EEPROM.
  - ❑ It will not be covered in this course.

6

6

User Defined Labels

- A user defined label is used to denote the memory location (address) of an instruction or a data item, and can be used in instructions to reference the instruction or the data item.
- Examples:

.dseg

amount:.byte 2

.cseg

formula: inc r0

.dseg

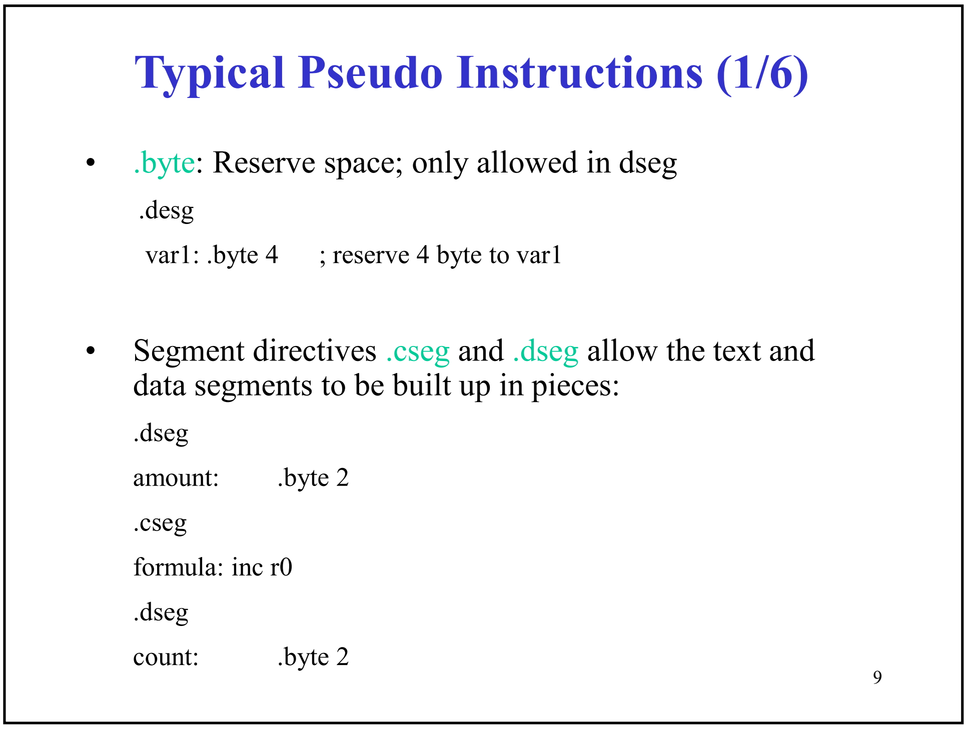
count: .byte 2

where **amount**, **formula**, and **count** are user defined labels. Note that there is a colon after a label.

Pseudo Instructions

- From AVR Studio Help
- These are for the AVR Studio Assembler

Directive	Description
BYTE	Reserve byte to a variable
CSEG	Code Segment
CSEGSIZE	Program memory size
DB	Define constant byte(s)
DEF	Define a symbolic name on a register
DEVICE	Define which device to assemble for
DSEG	Data Segment
DW	Define Constant word(s)
ENDM, ENDMACRO	End macro
EQU	Set a symbol equal to an expression
ESEG	EEPROM Segment
EXIT	Exit from file
INCLUDE	Read source from another file
LIST	Turn listfile generation on
LISTMAC	Turn Macro expansion in list file on
MACRO	Begin macro
NOLIST	Turn listfile generation off
ORG	Set program origin
SET	Set a symbol to an expression



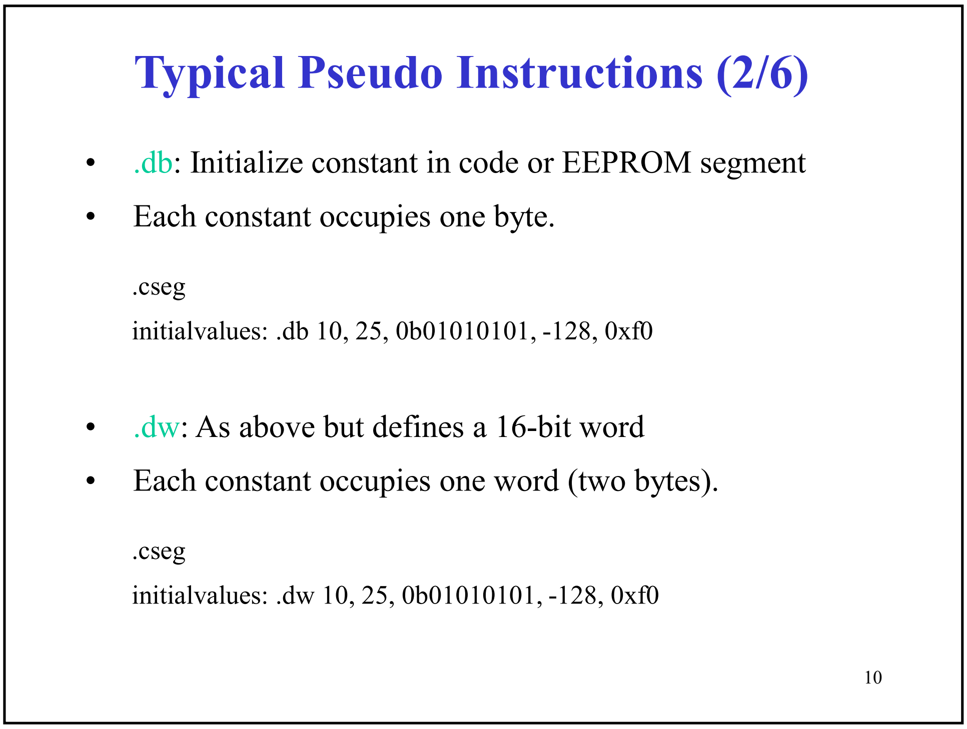
Typical Pseudo Instructions (1/6)

- .byte: Reserve space; only allowed in dseg

```
.desg
 var1: .byte 4      ; reserve 4 byte to var1
```

- Segment directives .cseg and .dseg allow the text and data segments to be built up in pieces:

```
.dseg
amount:      .byte 2
.cseg
formula: inc r0
.dseg
count:       .byte 2
```



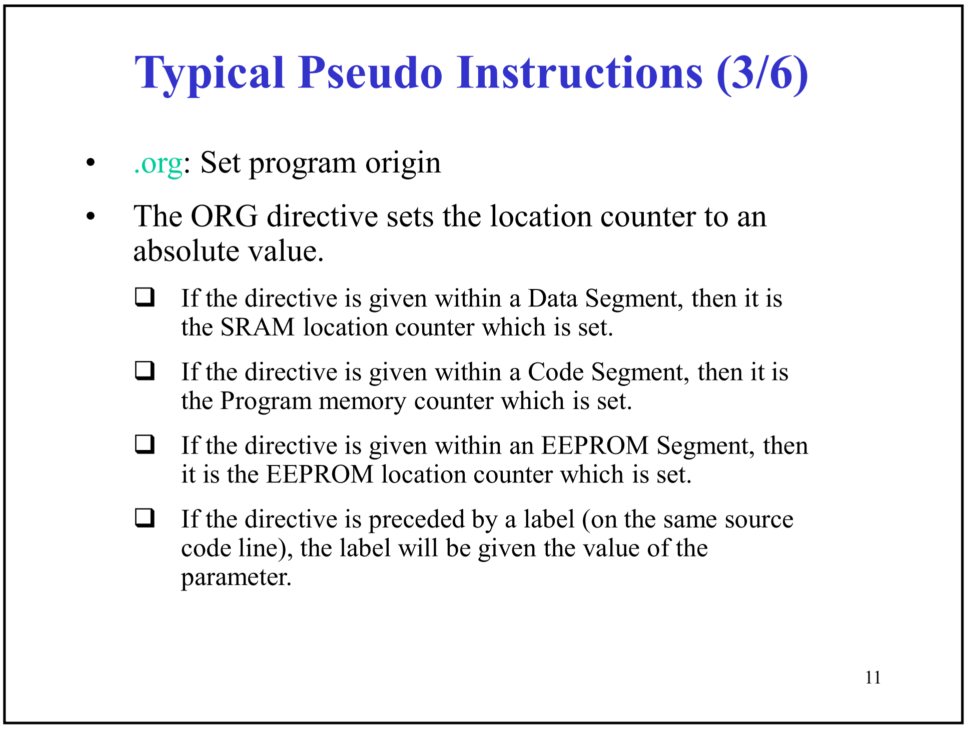
Typical Pseudo Instructions (2/6)

- .db: Initialize constant in code or EEPROM segment
- Each constant occupies one byte.

```
.cseg
initialvalues: .db 10, 25, 0b01010101, -128, 0xf0
```

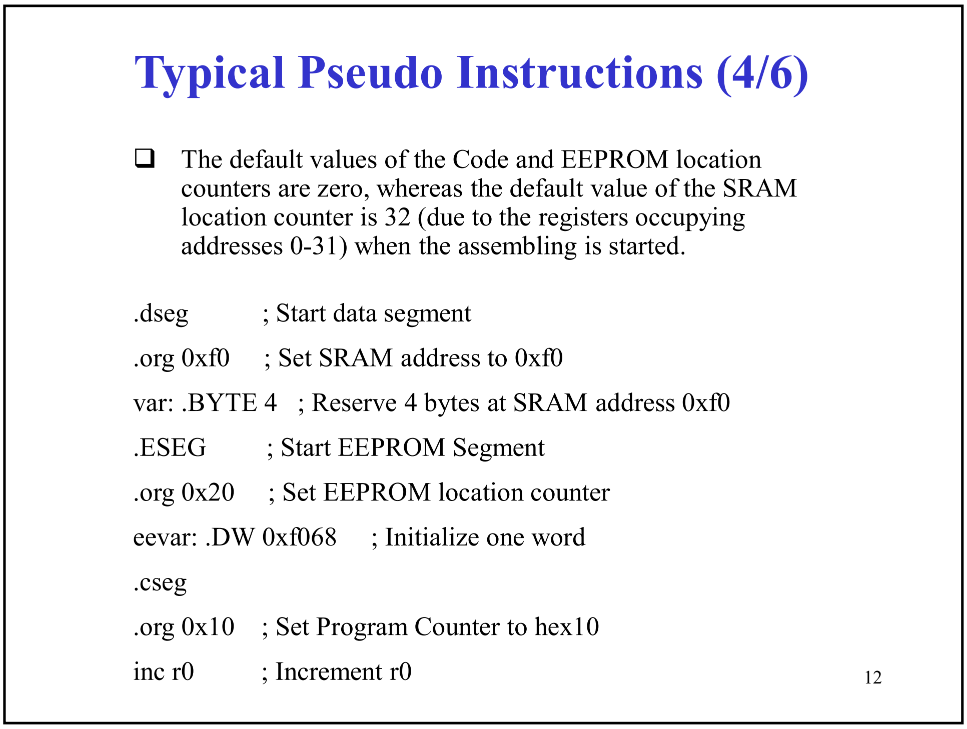
- .dw: As above but defines a 16-bit word
- Each constant occupies one word (two bytes).

```
.cseg
initialvalues: .dw 10, 25, 0b01010101, -128, 0xf0
```



Typical Pseudo Instructions (3/6)

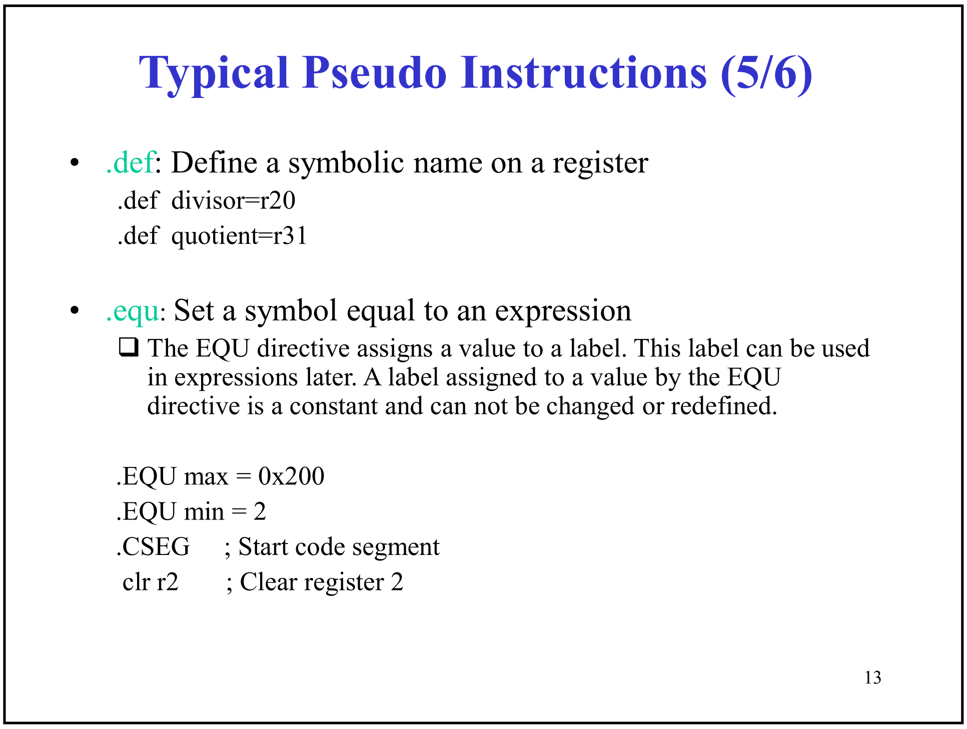
- .org: Set program origin
- The ORG directive sets the location counter to an absolute value.
  - ❑ If the directive is given within a Data Segment, then it is the SRAM location counter which is set.
  - ❑ If the directive is given within a Code Segment, then it is the Program memory counter which is set.
  - ❑ If the directive is given within an EEPROM Segment, then it is the EEPROM location counter which is set.
  - ❑ If the directive is preceded by a label (on the same source code line), the label will be given the value of the parameter.



Typical Pseudo Instructions (4/6)

❑ The default values of the Code and EEPROM location counters are zero, whereas the default value of the SRAM location counter is 32 (due to the registers occupying addresses 0-31) when the assembling is started.

```
.dseg         ; Start data segment
.org 0xf0     ; Set SRAM address to 0xf0
var: .BYTE 4   ; Reserve 4 bytes at SRAM address 0xf0
.ESEG         ; Start EEPROM Segment
.org 0x20     ; Set EEPROM location counter
eevar: .DW 0xf068     ; Initialize one word
.cseg
.org 0x10     ; Set Program Counter to hex10
inc r0        ; Increment r0
```



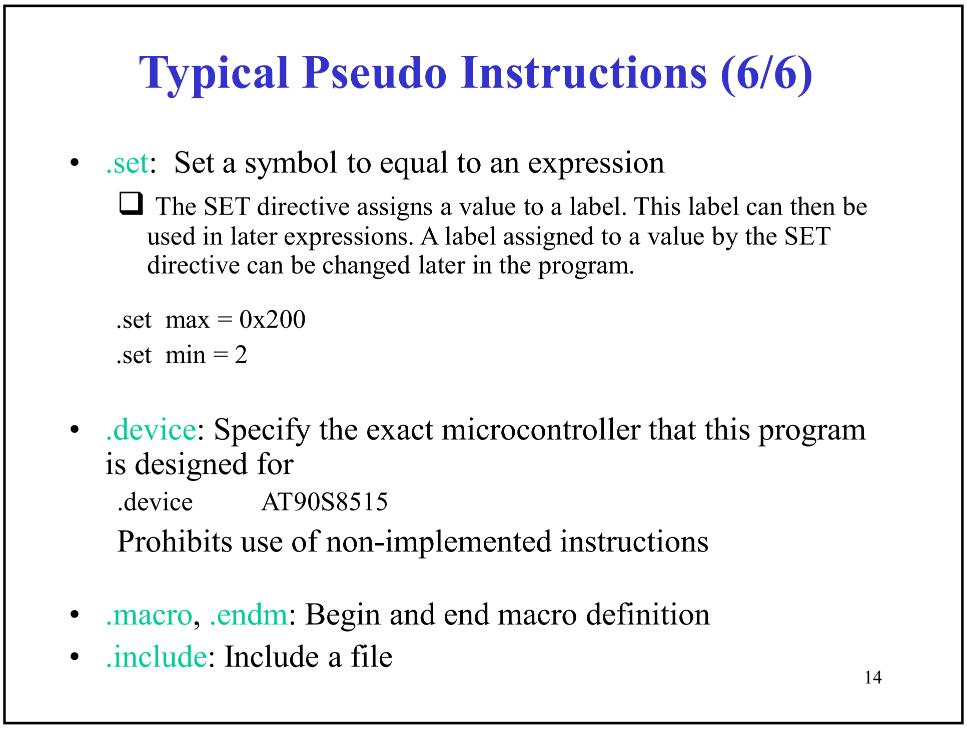
Typical Pseudo Instructions (5/6)

- .def: Define a symbolic name on a register

```
.def  divisor=r20
.def  quotient=r31
```

- .equ: Set a symbol equal to an expression
  - ❑ The EQU directive assigns a value to a label. This label can be used in expressions later. A label assigned to a value by the EQU directive is a constant and can not be changed or redefined.

```
.EQU max = 0x200
.EQU min = 2
.CSEG     ; Start code segment
 clr r2    ; Clear register 2
```



Typical Pseudo Instructions (6/6)

- .set: Set a symbol to equal to an expression
  - ❑ The SET directive assigns a value to a label. This label can then be used in later expressions. A label assigned to a value by the SET directive can be changed later in the program.

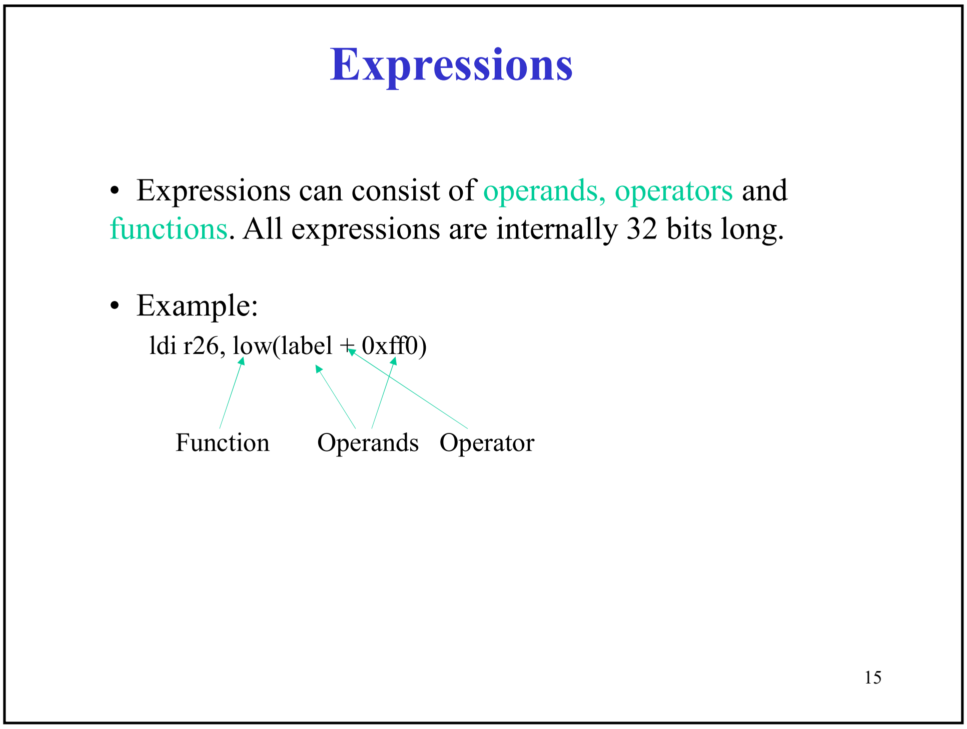
```
.set  max = 0x200
.set  min = 2
```

- .device: Specify the exact microcontroller that this program is designed for

```
.device         AT90S8515
```

  Prohibits use of non-implemented instructions

- .macro, .endm: Begin and end macro definition
- .include: Include a file



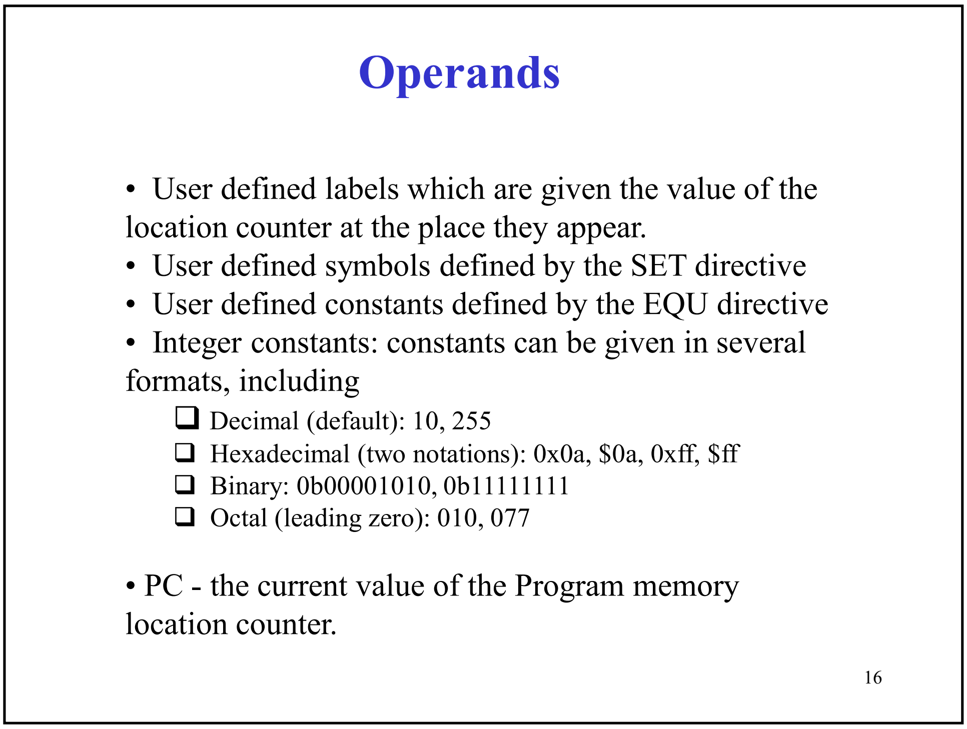
Expressions

- Expressions can consist of operands, operators and functions. All expressions are internally 32 bits long.

- Example:

```
ldi r26, low(label + 0xff0)
```

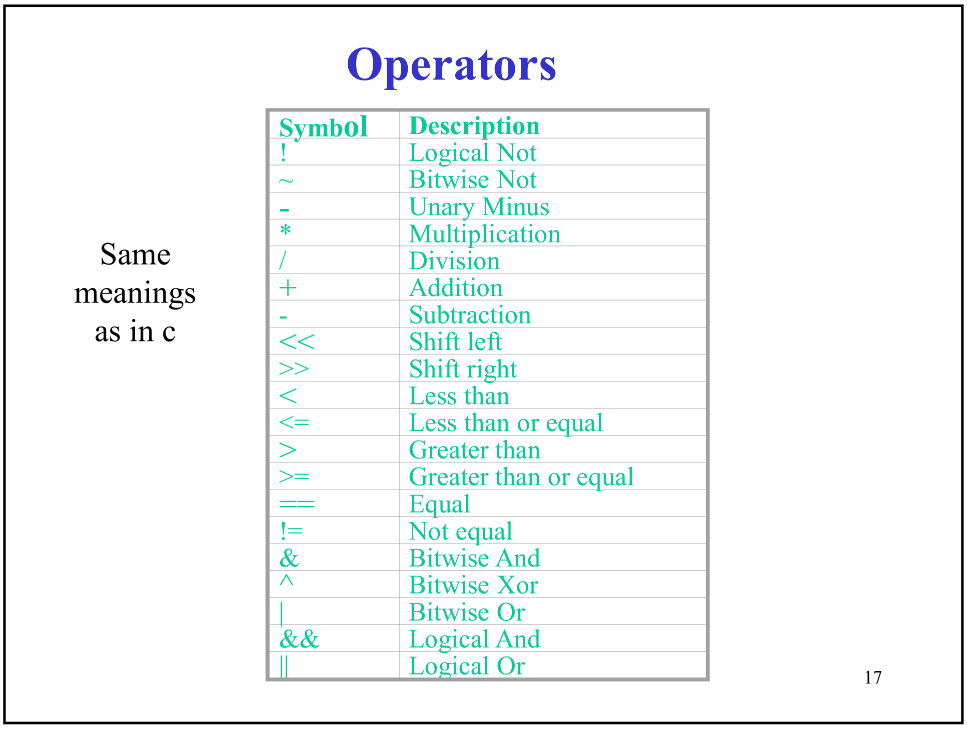
Function Operands Operator



Operands

- User defined labels which are given the value of the location counter at the place they appear.
- User defined symbols defined by the SET directive
- User defined constants defined by the EQU directive
- Integer constants: constants can be given in several formats, including
  - ❑ Decimal (default): 10, 255
  - ❑ Hexadecimal (two notations): 0x0a, $0a, 0xff, $ff
  - ❑ Binary: 0b00001010, 0b11111111
  - ❑ Octal (leading zero): 010, 077

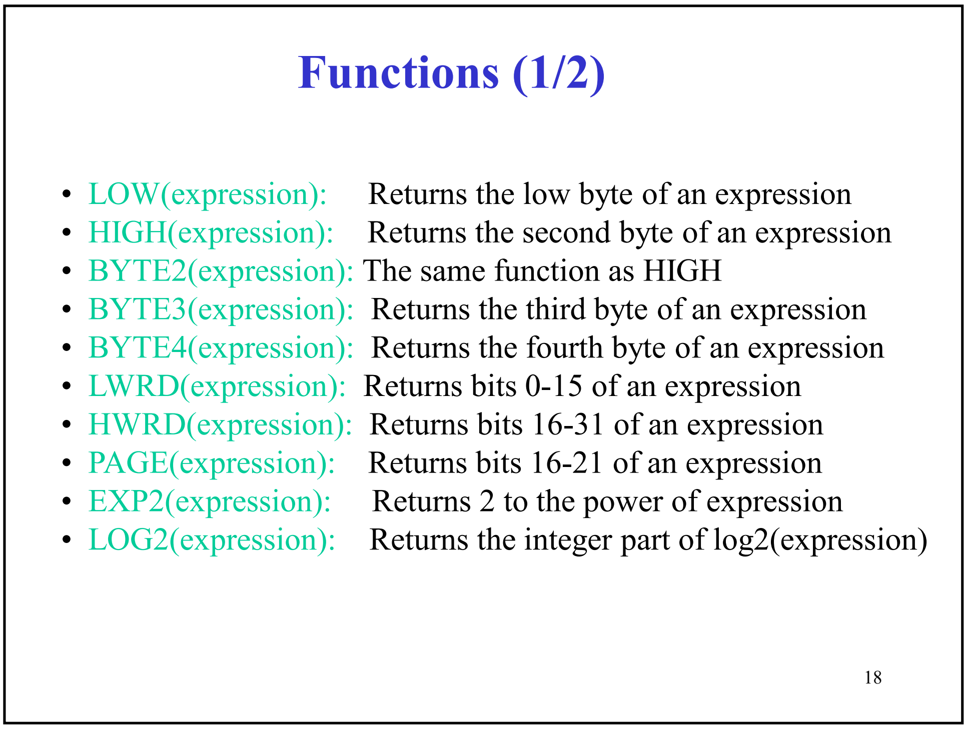
- PC - the current value of the Program memory location counter.



Operators

Same meanings as in c

Symbol	Description
!	Logical Not
~	Bitwise Not
-	Unary Minus
*	Multiplication
/	Division
+	Addition
-	Subtraction
<<	Shift left
>>	Shift right
<	Less than
<=	Less than or equal
>	Greater than
>=	Greater than or equal
==	Equal
!=	Not equal
&	Bitwise And
^	Bitwise Xor
|	Bitwise Or
&&	Logical And
||	Logical Or



Functions (1/2)

- LOW(expression): Returns the low byte of an expression
- HIGH(expression): Returns the second byte of an expression
- BYTE2(expression): The same function as HIGH
- BYTE3(expression): Returns the third byte of an expression
- BYTE4(expression): Returns the fourth 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 to the power of expression
- LOG2(expression): Returns the integer part of log2(expression)

Functions (2/2)

- Examples:

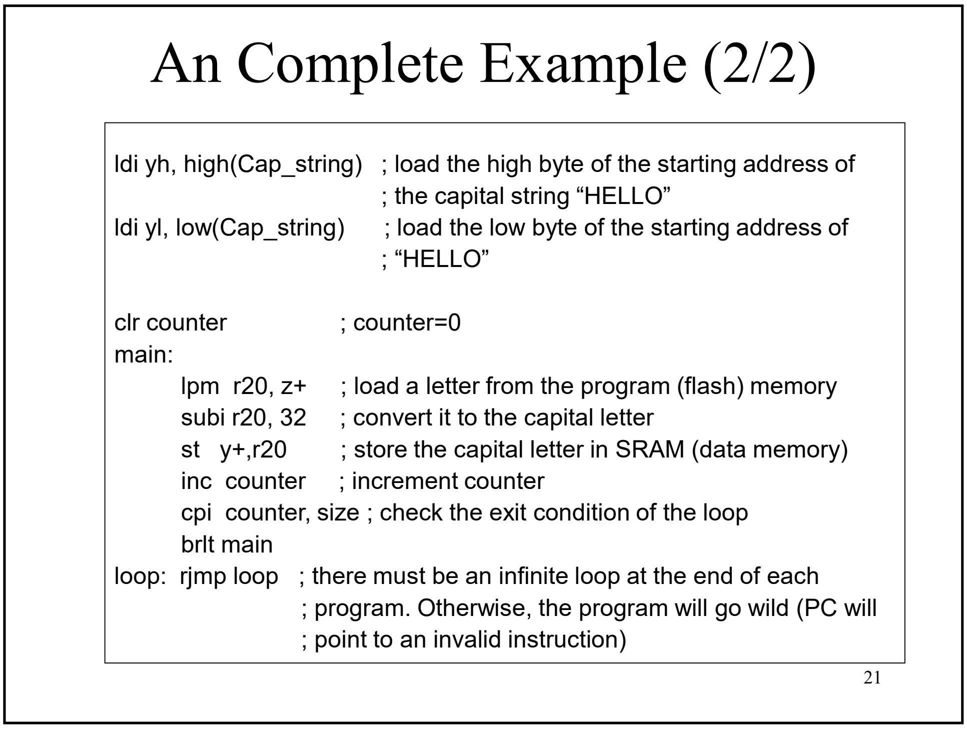
```
        cp   r0, low(-13167)
        cpc  r1, high(-13167)
        brlt  case1
         …
case1: inc r10
         …
```

An Complete Example (1/2)

```
; This program converts the string “hello” stored in the program memory
;  into the string “HELLO” stored in the data memory

.include "m64def.inc"
.equ size =5
.def counter =r17
.dseg    ; Data segment
.org 0x100    ; Set the starting address of data segment to 0x100
Cap_string: .byte 5   ; Allocate 5 bytes of data memory to store “HELLO”

.cseg      ; Code segment
Low_string: .db "hello“          ; “hello” is stored in the program memory
                  ldi zl, low(Low_string<<1)      ; load the low byte of
                                                   ; the address of "h“ into zl
                  ldi zh, high(Low_string<<1)    ; load the high byte of
                                                   ; the address of "h“ into zh
```

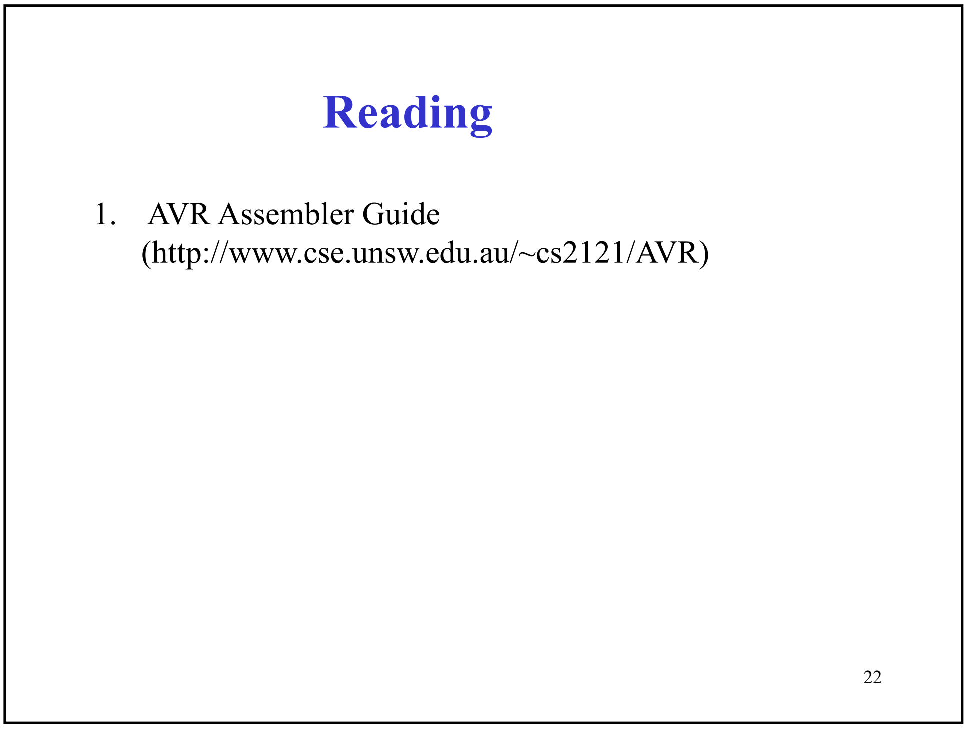


An Complete Example (2/2)

```
ldi yh, high(Cap_string)   ; load the high byte of the starting address of
                           ; the capital string “HELLO”
ldi yl, low(Cap_string)     ; load the low byte of the starting address of
                           ; “HELLO”

clr counter                ; counter=0
main:
        lpm  r20, z+       ; load a letter from the program (flash) memory
        subi r20, 32       ; convert it to the capital letter
        st   y+,r20        ; store the capital letter in SRAM (data memory)
        inc  counter       ; increment counter
        cpi  counter, size ; check the exit condition of the loop
        brlt main
loop:  rjmp loop    ; there must be an infinite loop at the end of each
                    ; program. Otherwise, the program will go wild (PC will
                    ; point to an invalid instruction)
```

21



Reading

1. AVR Assembler Guide (http://www.cse.unsw.edu.au/~cs2121/AVR)

22