CSSE2010

CSSE2010/CSSE7201
Lecture 20

Assembly

School of Information Technology and Electrical Engineering
The University of Queensland

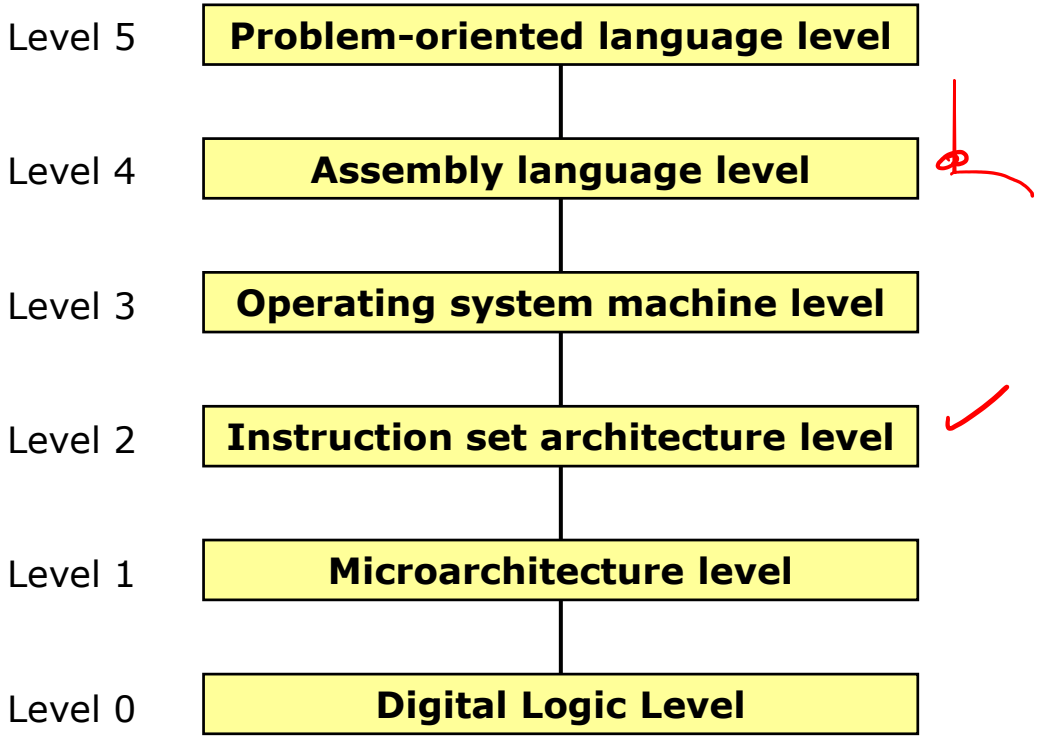
Admin

- Quiz 9 is due on Friday of this week
- Assignment 2 will be released Tuesday
  - CSSE7201 students will have an additional theoretical questions part which will be released separately and need to be submitted separately

Semester Break							
10	Mon-Tue	04-05 Oct	18	Serial I/O - Pre-recored due to public holiday on 4 Oct	15	AVR Interrupts	Quiz 8 (4pm, 8 Oct)
	Wed-Fri	06-08 Oct	19	More on Input-Output on AVR (6 Oct)	16	Serial I/O and ADC	
11	Mon-Tue	11-12 Oct	20	Assembly Process (11 Oct)	17	SPI/Communication, Assignment 2 released	Quiz 9 (4pm, 15 Oct)
	Wed-Fri	13-15 Oct	21	Compilation and Linking Process (13 Oct)		Assignment 2	
12	Mon-Tue	18-19 Oct	22	Memory and Disks (18 Oct)			Quiz 10 (4pm, 22 Oct)
	Wed-Fri	20-22 Oct	23	File System and Busses (20 Oct)			
13	Mon-Tue	25-26 Oct	24	Floating Point Numbers (25 Oct)			Assignment 2 (4pm, 1 Nov)
	Wed-Fri	27-29 Oct	25	Final Exam Review (27 Oct)			
Revision week - a 2-hour exam review session might be held							
Final Exam during the examination period, 2 hour invigilated exam, IN: paper-based on campus exam, EX: Proctor U online exam							

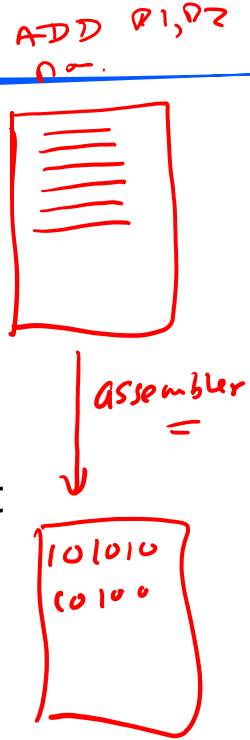
From Lecture 1

Structured Computer Organisation



Previously … Today…

- Previously
  - Many assembly language and instruction examples
  - The binary equivalent of the instruction – machine code.
- Today
  - **Assembly language** – more than instructions
  - **Assembly** - How does an assembly language program get turned into machine code?



Assembly Language Format

```
formula:        ldi r20, i          ; register r20 = i
```

- Statements have 4 parts (called fields):
  - a field for label
  - a field for operation (opcode)
  - a field for operand(s)
  - a field for comments
- Labels needed so statements can be jumped to, and so that data can be referenced
- A label usually starts in column 1
- Some assemblers require a semicolon to indicate end of a statement, others don't

Assembly Language Format

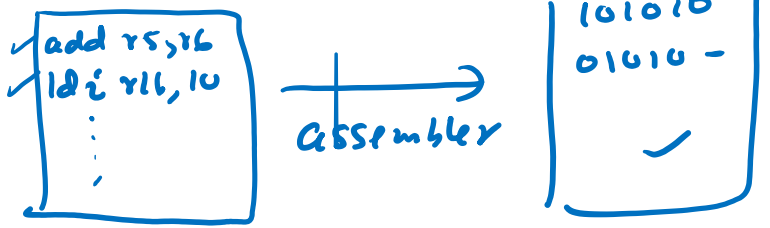
- Opcode has symbolic abbreviation for an instruction opcode
  - e.g. LDS for Load Direct from Data Space
- Operand field specifies addresses and registers used as operands of the instruction
- Register names vary from machine to machine
  - Pentium has eax, ebx, esi, edi, etc
  - Sparc has o0 .. o7, i0..i7, l0..l7, g0..g7
  - AVR has simple r0-r31
- Comments field is space for programmer to put helpful explanations
  - Only for human consumption
  - Vital for assembly language programs

Assembly Language Format

- What else do we need?
  - We need to provide other information to the assembler, e.g.
    - Where in memory the program should be put, etc.
    - Nicer names to use for registers (e.g. temp, SPH)
- A command (directive) to the assembler itself!
  - Example: `.byte`
- Precede with "."
- Called "**Pseudo-instruction**" or "**directive**"

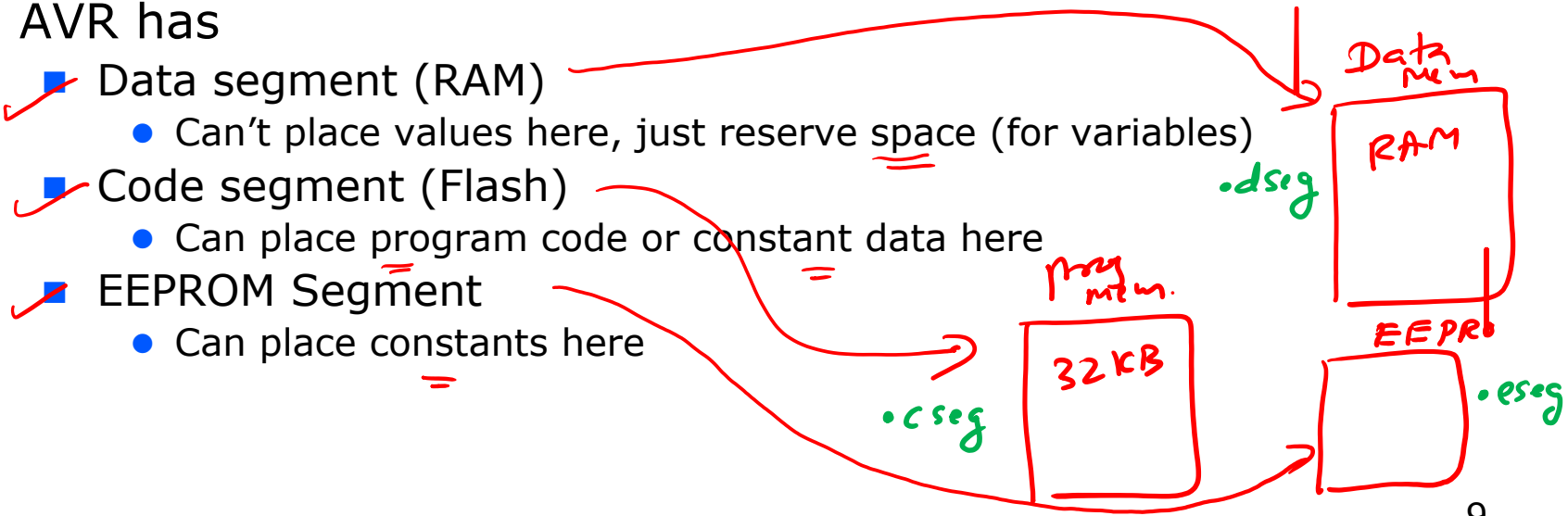
commands for the assembler.

.def temp=r16



Memory Segments

- Different types of memory are known as **segments** to the assembler
- Assembler directives enable code/data to be placed into different segments
- AVR has
  - Data segment (RAM)
    - Can't place values here, just reserve space (for variables)
  - Code segment (Flash)
    - Can place program code or constant data here
  - EEPROM Segment
    - Can place constants here



Pseudo-instructions (Directives)

- From Atmel Studio Help
  - There are more…
- These are for the Microchip Studio Assembler (AVRASM2)

Directive	Description
BYTE	Reserve byte(s) to a variable.
CSEG	Code Segment
CSEGSIZE	Program memory size
DB	Define constant byte(s)
DEF	Define a symbolic name on a register
DSEG	Data Segment
DW	Define Constant word(s)
ENDM, ENDMACRO	EndMacro
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
ELSE,ELIF	Conditional assembly
ENDIF	Conditional assembly

Pseudo-instructions

- `.byte`: Reserve space; only allowed in dseg
- Segment directives `.cseg` and `.dseg` allow the text and data segments to be built up in pieces:

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

- `.db`: Initialise constant in code or EEPROM segment
- `.dw`: As above but defines a 16-bit word

Pseudo-instructions (cont.)

- `.def`: Make a definition (for registers only)

```
.def   ZH=r31
.def   ZL=r30
```

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

```
.device ATmega324A
```

  Prohibits use of non-implemented instructions

- `.include`: Include a file
- `.exit`: Stop processing this file

Pseudo-instructions (cont.)

- `.equ`: Equate (not changeable)
- `.set`: Equate (but changeable)

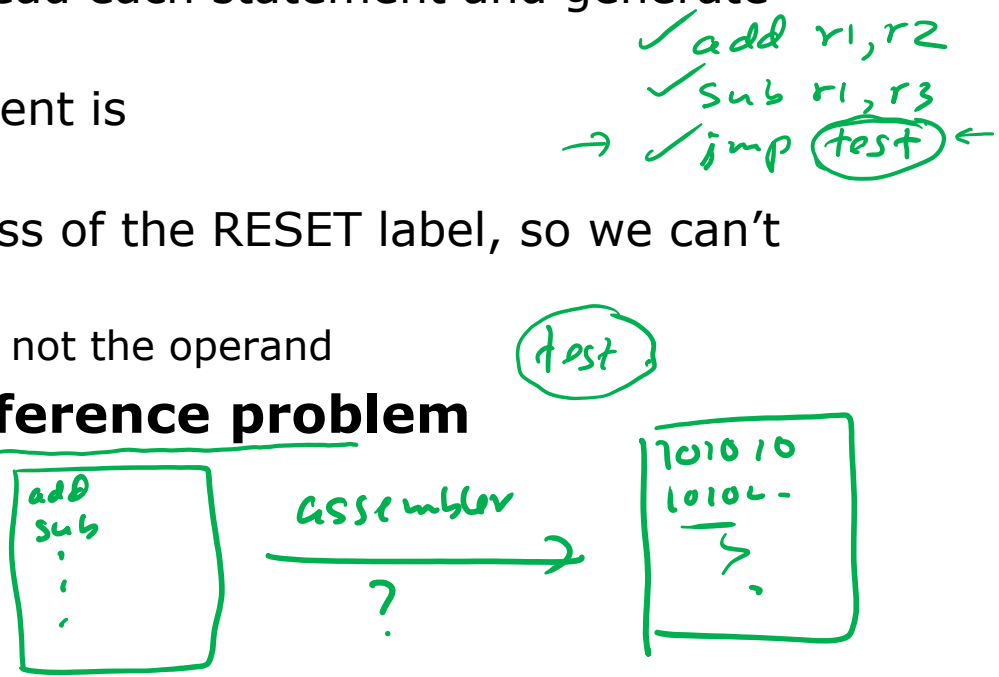
```
.equ sreg   = 0x3f    ; Status register
.set io_offset = 0x23 ; For now
```

- `.org`: Set location counter – i.e. address (in any segment)

- The above all apply to the assembler built into the Atmel Studio tool.
- Other assemblers (even targeting the AVR) use different syntax

Assembly Process

- Assembly language program consists of one line statements
  - We can not always just read each statement and generate machine code
  - For example, first statement is
    - `jmp RESET`
  - We don't know the address of the RESET label, so we can't generate the instruction
    - We know the opcode but not the operand
- This is called **forward reference problem**



Two Pass Assembly Process

- We need to process the file twice
- Pass One
  - Define all the symbols (labels etc)
- Pass Two
  - Values of symbols known
  - Can **assemble** each instruction
    - i.e. produce the actual machine code

Two Pass Assembly Example

cseg → cell size 16 bits

dseg → cell size 8 bits

```
                .equ    RAMEND = $08FF
                .equ    SPH = $3E
                .equ    SPL = $3D
                .def    ZH = r31
                .def    ZL = r30
0 →             jmp     RESET
                .dseg
256   varl:     .byte   2
258   var2:     .byte   2
                .cseg
                .def    temp = r16
2 →   mesg:     .db     72,101,108,108,111,0
5 →   table:    .dw     var1, var2
7 →   RESET:    ldi     temp, low(RAMEND)
                out     SPL, temp
                ldi     temp, high(RAMEND)
                out     SPH, temp
                …
                ldi     ZH, high(mesg*2)
                ldi     ZL, low(mesg*2)
                lpm     ;R0 <= M[Z]
```

Location counters :

- cseg : ~~0~~ ~~2~~ ~~3~~ ~~4~~ ~~5~~ ~~6~~ 7
- dseg : ~~256~~ ~~258~~ 260

Symbol table :

Symbols	Segment	Value
RAMEND	—	0x 8FF
SPH	—	0x 3E
:		
RESET	cseg	7
var1	dseg	256
var2	dseg	258
temp	—	r16
mesg	cseg	2
table	cseg	5

At what program address will the instruction at label RESET be placed?

- 5% A. 9
- 90% B. 12
- 5% C. 15
- 0% D. 18
- 0% E. None of the above

```
        0   jmp RESET
        2   jmp ISR1
        4   jmp ISR2
.dseg
var1:  .BYTE 1
var2:  .BYTE 3
var3:  .BYTE 2
.cseg
.def temp=r20
message1:  .DB 72,101,111,111,0,0
message2:  .DB 0,65,66,68,0,0
RESET:
        ldi temp, low(RAMEND)
        out SPL, temp
        ldi temp, high(RAMEND)
        out SPH, temp
```

0