CO 2103 Introduction to Assembly Language

Ong Wee Hong

owh@ieee.org, IPIHOAS G.38 or 2.34

Universiti Brunei Darussalam

Acknowledgement

The content of the slides used in this course are extracted from various sources including those quoted under references in following slides, teaching material from UBD lecturers who had taught this course before and from the material received from the author's course of studying

References - 1

- Books
  - Assembly language : step-by-step by Jeff Duntemann
  - Assembly language for Intel-based computers by Kip R. Irvine
  - Structured Computer Organization by Andrew S. Tanenbaum
  - How Computers Work by Ron White, Downs, Timothy Edward
  - How computers really work by Milind S. Pandit
  - IBM microcomputer assembly language : beginning to advanced by Godfrey J. Terry

References - 2

- Web pages – assembly language programming
  - Assembly language from Wikipedia
    - http://en.wikipedia.org/wiki/Assembly_language
  - The Place on the Internet to Learn Assembly at Webster
    - http://webster.cs.ucr.edu/
  - Assembly Language at OSdata.com
    - http://www.osdata.com/topic/language/asm/asmintro.htm

References - 3

- Web pages – assembly language programming
  - The Art of Assembly Language Programming by Randall Hyde
    - http://webster.cs.ucr.edu/AoA/DOS/AoADosIndex.html
  - Complete 8086 instruction set
    - http://www.emu8086.com/assembly_language_tutorial_assembler_reference/8086_instruction_set.html
  - documentation for 8086 assembler and emulator
    - http://www.emu8086.com/assembly_language_tutorial_assembler_reference/

References - 4

- Web pages - hardware
  - About CPUs at Karbosguide.com
    - http://www.karbosguide.com/hardware/module3a1.htm
  - Birth of a Chip by Linley Gwennap
    - http://www.byte.com/art/9612/sec6/art2.htm
  - Chronology of Personal Computers by Ken Polsson
    - http://www.islandnet.com/~kpolsson/comphist/

References - 5

- Web pages – useful reading
  - flat assembler
    - http://flatassembler.net/
  - Unix Assembly Language Programming by G Adam Stanislav
    - http://www.int80h.org/
  - PowerPC Assembly Programming on the Mac Mini by Pramode C.E
    - http://linuxgazette.net/117/pramode.html
  - PIC Assembly Language for the Complete Beginner
    - http://www.ai.uga.edu/mc/microcontrollers/pic/picassem2004.pdf

Course Content

- Introduction
- Background Knowledge
  - Digital Logic, Data Representation
- Intel 8086 Microprocessor
- 8086 Assembly vs Machine Language
- Basics of 8086 Assembly Language Programming
- More into Assembly Language Programming …

Course Management

- Learning activities (subject to alternative arrangements)
  - Lectures, laboratories and tutorials
    - 10:10-12:00am, FSM 1.21, Tuesday
    - 10:10-12:00am, FSM 1.19, Saturday
- Assessment scheme
  - Examination 70% - 120 min exam in Nov/Dec
  - Coursework 30% - few problem solving

What is Assembly Language (AL)?

- Machine-specific programming language
  - an assembly language is a low-level language for programming computers. It implements a symbolic representation of the numeric machine codes and other constants needed to program a particular CPU architecture. This representation is usually defined by the hardware manufacturer, and is based on abbreviations (called mnemonics) that help the programmer remember individual instructions, registers, etc. An assembly language is thus specific to a certain physical or virtual computer architecture (as opposed to most high-level languages, most of which are portable). [quoted from Wikipedia]

Why learn AL?

- Computers don’t understand our languages; neither Java, C, Pascal, etc
- Someone has got to know their languages to be able to ask them to work
- Gain insight into hardware concepts and learn how a processor works
- Direct control over hardware for efficiency
- To program embedded systems
- More …

Why not just AL?

- After completion of this course, try code one of the large program you have created in Java (or other language) in AL and you will have the answer to the above question
- Not portable, i.e. processor-specific
- Normally include AL in HLL. When additional performance is required for high-level languages (HLL), AL can enhance the performance of these languages with small, fast and powerful AL code modules. This allows the HLL to target critical areas of their code in a very efficient and convenient manner.

AL vs HLL

Assembly Language	High Level Language
Not intuitive (c.f. English)	Intuitive (English like)
Uses instructions specific to the processor	Uses commands and rules of compiler (C++, Java, etc)
1-to-1 correspondence to Machine Language ML (zeros and ones) – direct conversion	Each command may be converted to few AL instructions before converting to ML
Knowledge of architecture of processor essential to program in AL	Knowledge of processor architecture not required to program in HLL
Not portable (specific processor)	Portable (any processor)
Can be efficient and small in size	Easier to program and shorter program (less HLL command lines)

Sample AL Program (8086)

```
Start proc
    mov     ax,@data
    mov     ds,ax
    mov     es,ax
    mov     [SegCode],ax

    mov     ax,0013h    ;changes to 320x200x256 graphics mode
    int     10h

    call    WriteLines
MainLoop:

    mov     dx,3dah
VRT:
    in      al,dx
    test    al,8
    jnz     VRT         ;wait until Verticle Retrace starts

    call    RotatePalette

    mov     dx,3dah
NoVRT:
    in      al,dx
    test    al,8
    jz      NoVRT       ;wait until Verticle Retrace ends
                        ;so that we dont rotate more than once a frame

    mov     ah,1        ;wait for a keypress
    int     16h
    jz      MainLoop

    mov     ah,0
    int     16h         ;get the key
    neg     cs:[PalIndexVel]
    cmp     al," "
    je      MainLoop

    mov     ax,0003h    ;changes to 80x25x16 text mode
    int     10h
    mov     ax,4c00h    ;terminate process and
    int     21h         ;return control to DOS

Start endp
end Start
```

Into Hardware …

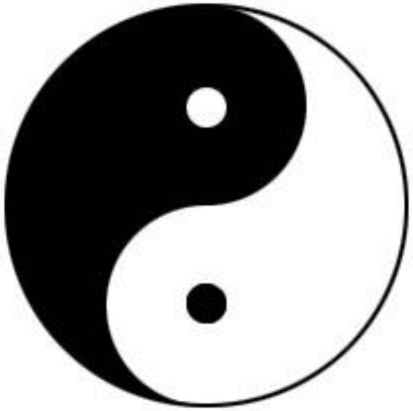
Before we are able to make sense out of what we have mentioned in previous slides and what we will be learning, we need to have some basic knowledge of hardware

Hardware and Software - 1

- Hardware
  - parts you can touch
    - CPU, keyboard, screen, circuit boards, wires, etc.
  - physical components
- Software
  - programs
    - consist of instructions telling the computer what to do
  - ability to run different programs makes the computer a General Purpose machine
  - abstract components

Hardware and Software - 2

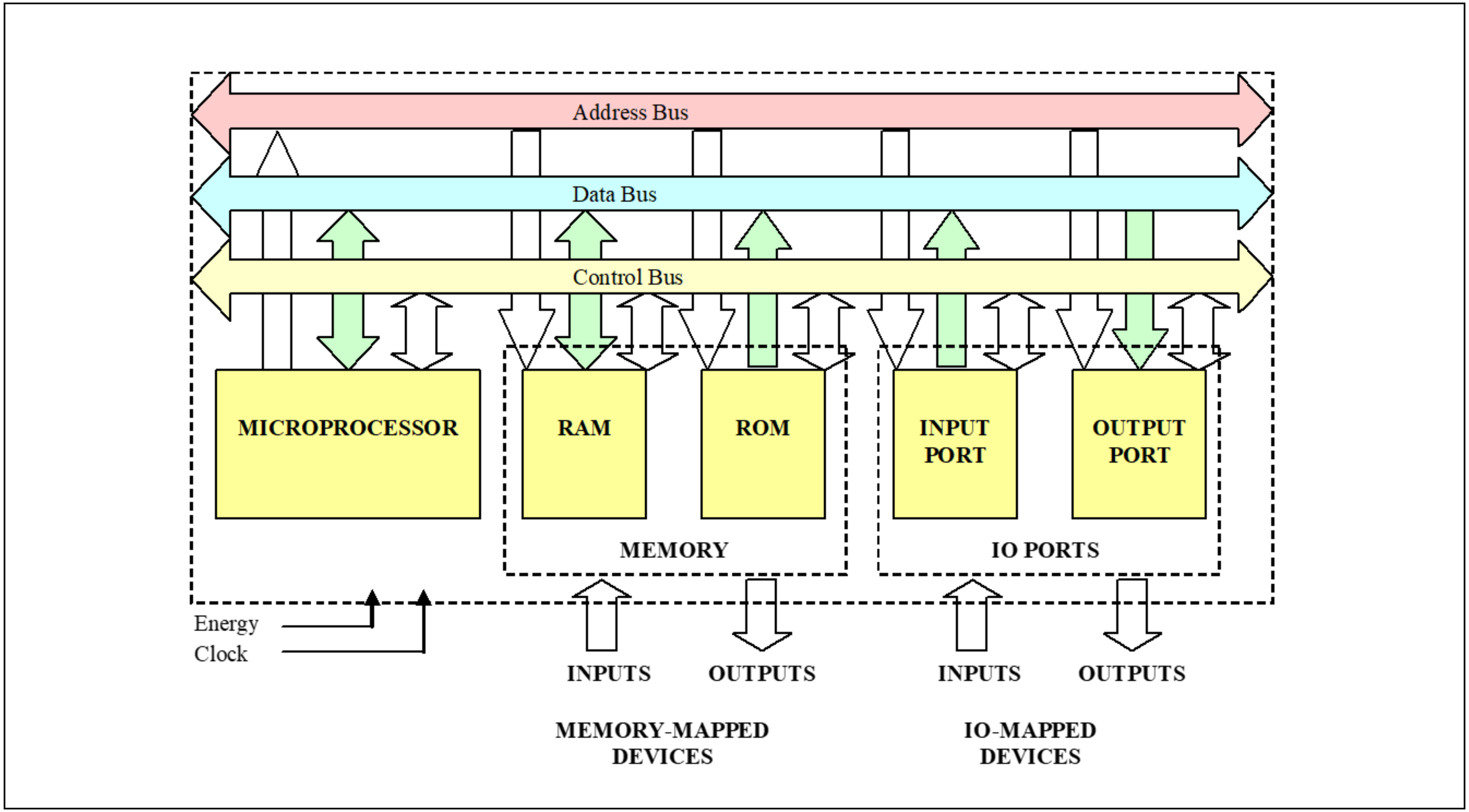
- Software without the hardware to execute is useless
- Software gives intelligence to the hardware



From the big picture ...

We will look at the hardware organization of a microcomputer or personal computer (PC) to begin with ...

General Microcomputer Organization - 1



General Microcomputer Organization - 2

- Microprocessor or Central Processing Unit (CPU)
  - can be considered as the brain of the system. It controls all activities within the system according to instructions given to it.
- Memory
  - RAM (Random Access Memory) – volatile read/write memory for storing information being processed
  - ROM (Read Only Memory) – non-volatile read only memory for storing system programs

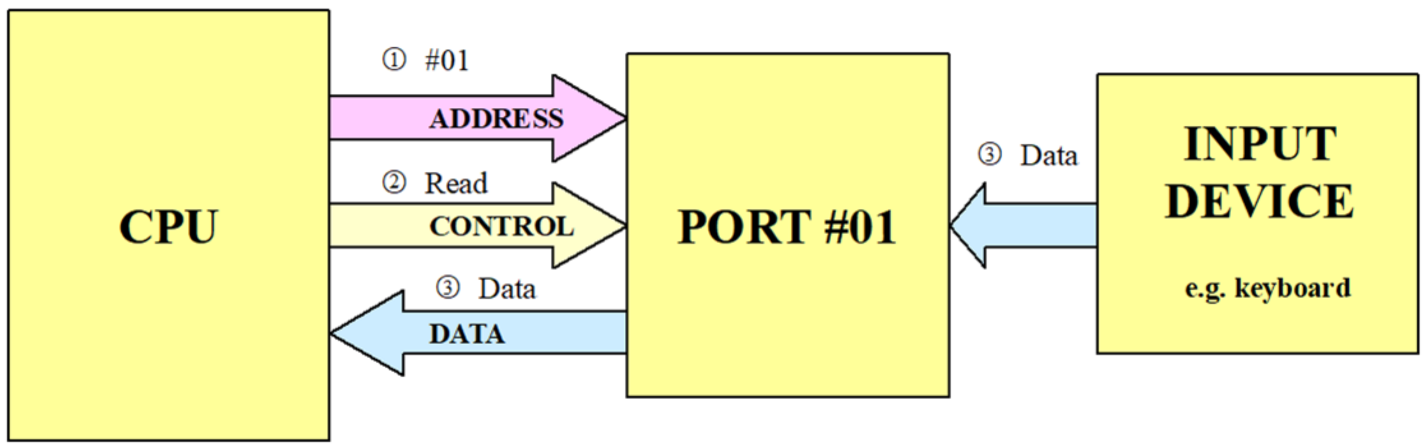
General Microcomputer Organization - 3

- IO Ports
  - Input Port - the point of the system where all external data/information enter the system. External input devices (e.g. keyboard) are connected to input ports.
  - Output Port - the point of the system where the data/information are sent to the external world. External output devices (e.g. printer) are connected to output ports.
  - IO-Mapped Devices use different address space from memory
  - Memory-Mapped Devices are external devices that make use of memory address space

General Microcomputer Organization - 4

- Buses
  - cable, group of wires, signals
  - Data Bus – where uP communicates data/information with other devices
  - Address Bus – where uP sends the address of a device to select it
  - Control Bus – where control signals (synchronization, IO signals, interrupt) are communicated between uP and devices

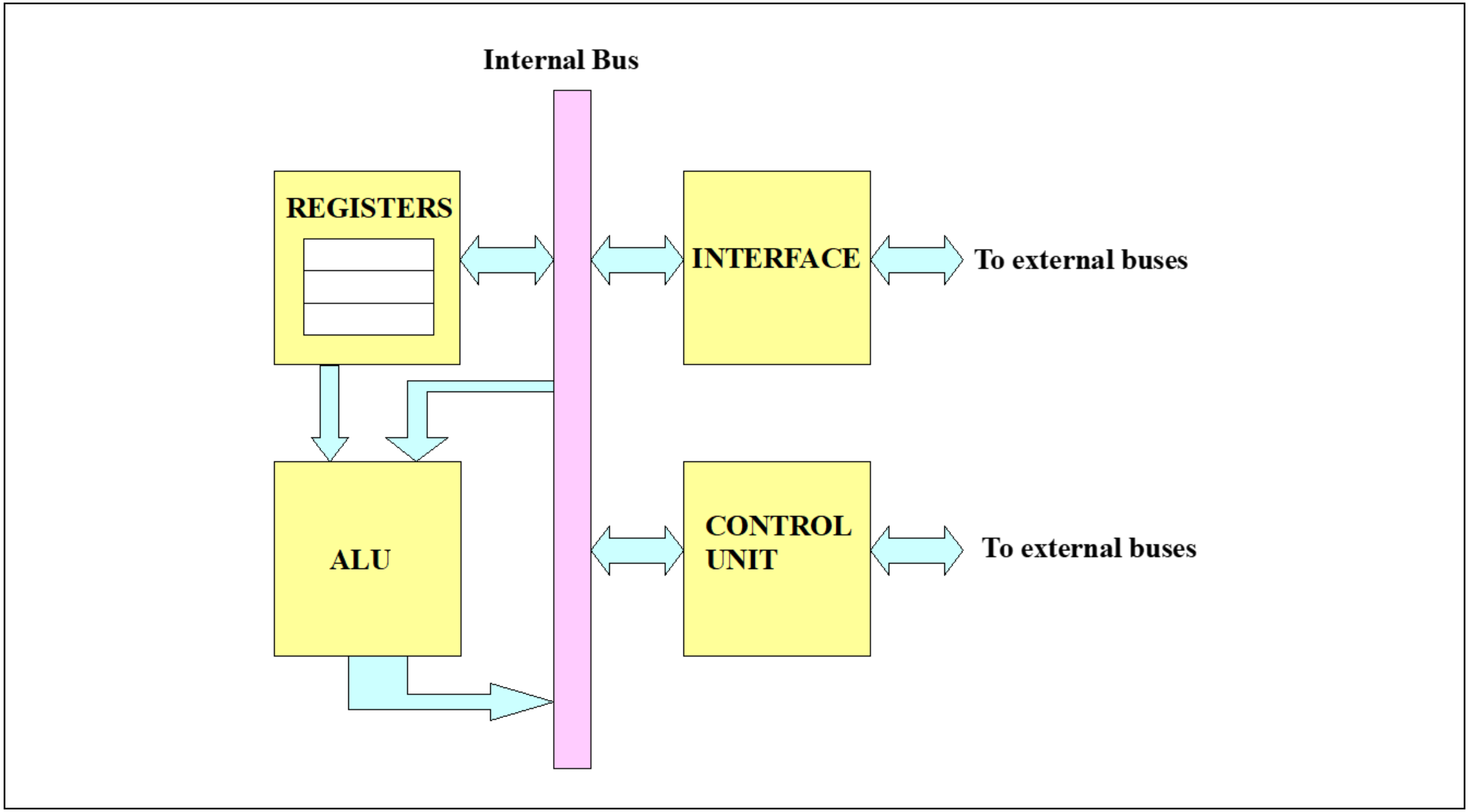
uComputer in Action – Read Data



Into the important part …

Knowing the general organization of the PC, we now look into what made up its CPU …

CPU Components - 1



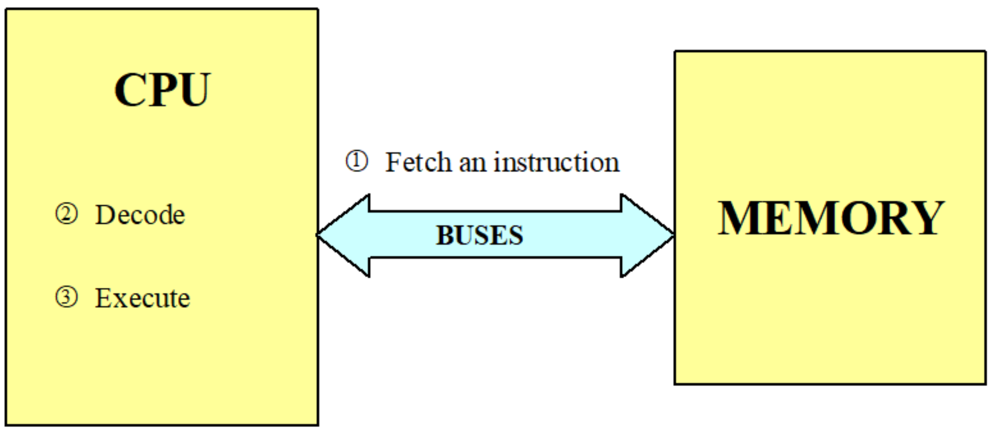
CPU Components - 2

- Register Array (RA) or Registers
  - provide fast storage for immediate processing
- Arithmetic and Logic Unit (ALU)
  - performs arithmetic (+, –, ×, ÷) and logic (NOT, OR, AND, etc) operations
- Control Unit (CU)
  - co-ordinates the different components of the CPU
- Interface
  - connects CPU signals to external devices (memory and IO)

CPU in Action - 1

- CPU does only one thing for all time that it is alive:
  - execute programs
- It can only be as useful as its programs, i.e. what it is programmed to do
- Program execution has 3 phases, called Fetch-Execute cycle:
  - Fetch an instruction – from memory
  - Decode the instruction – in Control Unit
  - Execute the instruction

CPU in Action - 2



Available CPU

- There are many different CPU designed by different companies
- John Bayko has compiled a list of "Great Microprocessors of the Past and Present"
  - http://jbayko.sasktelwebsite.net/cpu.html
- Examples are Intel 80x86, Intel P, Motorola 68k, Zilog Z-8k, PowerPC, Microchip PIC
- Different family has different details in their internal structure
- Different family uses different set of instructions, i.e. they speak different languages

Heading back to AL …

The brain of the PC is the CPU and there are many different CPU available in the market and they speak different languages …

Sample AL Programs - 1

Zilog Z80

```
.nolist
#include   "ti83plus.inc"
#define    ProgStart    $9D95
.list
.org    ProgStart - 2
    .db    t2ByteTok, tAsmCmp
    b_call(_ClrLCDFull)
    ld    hl, 0
    ld    (PenCol), hl
    ld    hl, msg
    b_call(_PutS)            ; Display the text
    b_call(_NewLine)
    ret
msg:
    .db "Hello world!", 0
.end
.end
```

Intel 8086

```
DATA SEGMENT
    MSG DB "Hello, World!","$"
DATA ENDS

CODE SEGMENT
    ASSUME CS:CODE, DS:DATA
START:
    MOV AX, DATA        ;INTILIZE
    MOV DS, AX

    MOV AH, 09H         ;PRINT STRING
    LEA DX, MSG
    INT 21H

STOP:
    MOV AX, 4C00H       ;TERMINATE
    INT 21H
CODE ENDS
END START
```

Sample AL Programs - 2

Motorola 6809

```
NAME hello_world
TITL 'This program prints  on TX16W'

* Some status and command codes and addresses for the LCD
lcd_clear =     $1
lcd_busy =      $80
lcd_instr =     $100000
lcd_status =    $100000
lcd_data =      $100002

 rseg main

* This is the programs entry point。
start:

 bsr    F_LCDclr
 pea    text(pc)
 bsr    F_LCDprint
 addq   #4,sp

* Eternal loop! This is where you have to turn off and reboot the TX16W.
suicide:
 bra    suicide

* This procedure initialise the LCD-screen
F_LCDclr:
 bsr.s      F_LCDwait
 move.w     #lcd_clear,lcd_instr
 rts
```

```
* This function waits until busy-flag goes low.
F_LCDwait:
 move.w     d0,-(a7)
wait_loop1:
 move.w     lcd_status,d0
 and.w      #lcd_busy,d0
 bne.s      wait_loop1

 move.w     (a7)+,d0
 rts

* This procedure prints a string to the current cursor position.
F_LCDprint:
 move.l     a0,-(a7)
 move.l     8(a7),a0
print_loop1:
 bsr    F_LCDwait
 move.b     (a0)+,d0
 beq.s      print_eos
 move.w     d0,lcd_data
 bra.s      print_loop1

print_eos:
 move.l     (a7)+,a0
 rts

text:
 dc.w        'Hello World',0

 end
```

Sample AL Programs - 3

Microchip PIC16f84

```
; Program to send "Hello World!" to a PC COM port

MSGTXT   ADDWF PCL, f      ; offset added to PCL
         RETLW $48         ; 'H'
         RETLW $65         ; 'e'
         RETLW $6C         ; 'l'
         RETLW $6C         ; 'l'
         RETLW $6F         ; 'o'
         RETLW $20         ; ' '
         RETLW $57         ; 'W'
         RETLW $6F         ; 'o'
         RETLW $72         ; 'r'
         RETLW $6C         ; 'l'
         RETLW $64         ; 'd'
         RETLW $21         ; '!'
         RETLW $0D         ; carriage return
         RETLW $0A         ; line feed
         RETLW $00         ; indicates end

OUTMSG   MOVWF MSGPTR      ; put 'W' into message pointer
MSGLOOP  MOVF MSGPTR, W    ; put the offset in 'W'
         CALL MSGTXT       ; returns ASCII character in 'W'
         ADDLW 0           ; sets the zero flag if W = 0
         BTFSC STATUS, Z   ; skip if zero bit not set
         RETURN            ; finished if W = 0
         CALL OUTCH        ; output the character
         INCF MSGPTR, f    ; point at next
         GOTO MSGLOOP      ; more characters
```

```
OUTCH    MOVWF TXREG       ; put W into transmit register
         MOVLW 8           ; eight bits of data
         MOVWF BITS        ; a counter for bits
         BSF PORTA, 2      ; start bit (flipped remember), RA2
TXLOOP   MOVLW $31         ; 49 decimal, delay time
         CALL MICRO4       ; wait 49 x 4 = 196 microseconds
         RRF TXREG, f      ; roll rightmost bit into carry
         BTFSC STATUS, C   ; if carry 0 want to set bit, ( a low )
         GOTO CLRBIT       ; else clear bit, ( a high )
         BSF PORTA, 2      ; +5V on pin 1 ( RA2 )
         GOTO TESTDONE     ; are we finished?
CLRBIT   BCF PORTA, 2      ; 0V on pin 1 ( RA2 )
         NOP               ; to make both options 12 micosec
TESTDONE DECFSZ BITS, f    ; 1 less data bit, skip when zero
         GOTO TXLOOP       ; more bits left, delay for this one
         MOVLW $34         ; full 208 microsec this time
         CALL MICRO4       ; delay for last data bit
         BCF PORTA, 2      ; 0V, ( a high ) for stop bits
         MOVLW $68         ; decimal 104 delay for 2 stop bits
         CALL MICRO4
         RETURN

MSEC1    MOVLW $F9         ; allow for 4 microsec overhead..
         NOP               ; (2 for CALL)
MICRO4   ADDLW $FF         ; subtract 1 from W
         BTFSS STATUS,Z    ; skip when you reach zero
         GOTO MICRO4       ; more loops
         RETURN
```

AL programs are hardware dependent

Sample Instruction Sets - 1

Motorola 6809 Instruction Set:

ABX - Add to Index Register
ADCa s - Add with Carry
ADDa s - Add
ADDD s - Add to Double acc
ANDa s - Logical AND
ANDCC s - Logical AND with CCR
ASL d - Arithmetic Shift Left
ASLa - Arithmetic Shift Left
ASR d - Arithmetic Shift Right
ASRa - Arithmetic Shift Right
BCC m - Branch if Carry Clear
BCS m - Branch if Carry Set
BEQ m - Branch if Equal
BGE m - Branch if Great/Equal
BGT m - Branch if Greater Than
BHI m - Branch if Higher
BHS m - Branch if Higher/Same
BITa s - Bit Test accumulator
BLE m - Branch if Less/Equal
BLO m - Branch if Lower
BLS m - Branch if Lower/Same
BLT m - Branch if Less Than

Intel 8086/80186/80286/80386/80486 Instruction Set:

AAA - Ascii Adjust for Addition
AAD - Ascii Adjust for Division
AAM - Ascii Adjust for Multiplication
AAS - Ascii Adjust for Subtraction
ADC - Add With Carry
ADD - Arithmetic Addition
AND - Logical And
ARPL - Adjusted Requested Privilege Level of Selector (286+ PM)
BOUND - Array Index Bound Check (80188+)
BSF - Bit Scan Forward (386+)
BSR - Bit Scan Reverse (386+)
BSWAP - Byte Swap (486+)
BT - Bit Test (386+)
BTC - Bit Test with Compliment (386+)
BTR - Bit Test with Reset (386+)
BTS - Bit Test and Set (386+)
CALL - Procedure Call
CBW - Convert Byte to Word
CDQ - Convert Double to Quad (386+)
CLC - Clear Carry
CLD - Clear Direction Flag
CLI - Clear Interrupt Flag (disable)
CLTS - Clear Task Switched Flag (286+ privileged)
CMC - Complement Carry Flag

Sample Instruction Sets - 2

Microchip PIC16f84 Instruction Set:

addlw k - Add literal to W
addwf f,d - Add W and f
andlw k - AND literal and W
andwf f,d - AND W and f
bcf f,b - Bit clear f
bsf f,b - Bit set f
btfsc f,b - Bit test, skip next instruction if clear
btfss f,b - Bit test, skip next instruction if set
call k - Call subroutine
clrf f - Clear f
clrw - Clear W
clrwdt - Clear watchdog timer
comf f,d - Complement f
decf f,d - Decrement f
decfsz f,d - Decrement f, skip if zero f
goto k - Goto address k
incf f,d - Increment f
incfsz f,d - Increment f, skip if zero
iorlw k - Incl. OR literal and W
iorwf f,d - Inclusive OR W and f
movf f,d - Move f
movlw k - Move Literal to W

Zilog Z80 Instruction Set:

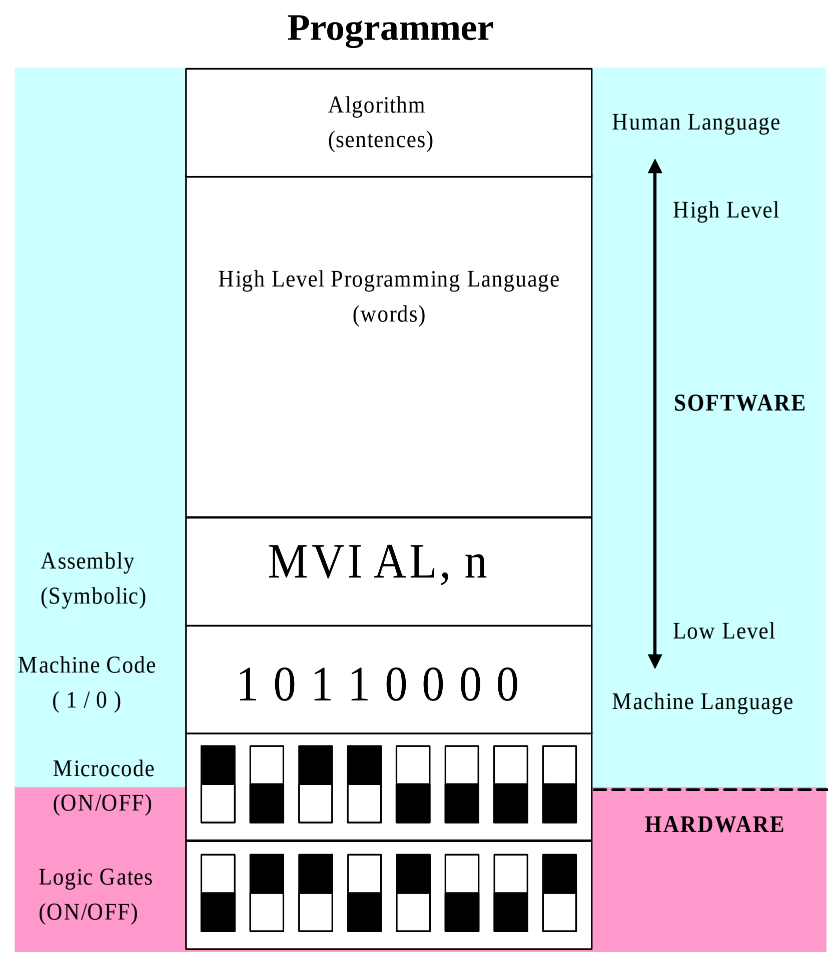
ADC	ADD WITH CARRY
ADD	ADD
AND	LOGICAL AND
BIT	BIT TEST
CALL	CALL SUB ROUTINE
CCF	COMPLEMENT CARRY FLAG
CP	COMPARE
CPD	COMPARE AND DECREMENT
CPDR	COMPARE DECREMENT AND REPEAT
CPI	COMPARE AND INCREMENT
CPIR	COMPARE INCREMENT AND REPEAT
CPL	COMPLEMENT ACCUMULATOR
DAA	DECIMAL ADJUST ACCUMULATOR
DEC	DECREMENT
DI	DISABLE INTERRUPTS
DJNZ	DEC JUMP NON-ZERO
EI	ENABLE INTERRUPTS
EX	EXCHANGE REGISTER PAIR
EXX	EXCHANGE ALTERNATE REGISTERS
HALT	HALT, WAIT FOR INTERRUPT OR RESET
IM	INTERRUPT MODE 0 1 2
IN	INPUT FROM PORT
INC	INCREMENT

AL Bad and Good News

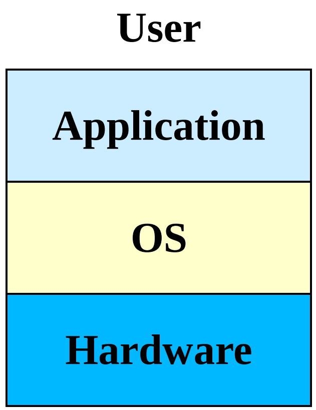
- Bad news
  - AL is machine-specific (or CPU-specific)
  - There are many different CPU available
  - Too heavy to learn AL for all of them
- Good news
  - We will only learn AL for Intel 80x86 CPU
  - It’s easy to learn the others once knowing one

Levels of Abstraction - 1

- Generalize a computer into different levels
- From the programmer's point of view:
  - High-Level Programming Languages
  - Assembly Language
  - Machine Code
  - Microcode (CISC)
  - Logic Gates



Levels of Abstraction - 2



- From the User's Point-of-View:
  - Applications Software
    - Word Processor, Spreadsheet, etc.
  - Operating System Software
    - UNIX, MS-DOS, OS/2, VMS, etc.
  - Hardware
    - Mainframe, Workstation, Personal Computer
- Applications are written for a specific Operating System
  - Operating System shields the Application from the Hardware
  - Different combinations of Hardware Platform, Operating System and Applications are possible

High Level vs Low Level Programming

- Working at higher levels …
  - Programming is easier
  - Programs are more portable (hardware independent)
  - Little or no knowledge of hardware required
- Working at Lower Levels …
  - More control over the machine
  - Unrestricted access to hardware
  - Requires specific knowledge of target hardware
  - Possible to write small, very efficient programs

Translation Programs - 1

- Rationale
  - CPU is an electronic device (hardware)
  - CPU only understands electronic signals
  - CPU uses ON (1) or OFF (0) as electronic signals; called logic signals
  - CPU only understands 0 and 1
  - Native language of all CPU is the Machine Language (Machine Code) made up of string of 0s and 1s
  - Programs written, not in Machine Codes, must be translated into Machine Codes for storage and for the CPU to execute

Translation Programs - 2

- Compiler
  - Translates High-level language (HLL) file to file of Machine Code instructions
  - HLLs are independent of CPU type
  - Examples of compiled languages: C/C++, Pascal, FORTRAN
- Assembler
  - Translates Assembly Language file to Machine Code file
  - Assembly Language is specific to CPU type
  - Assembly Language is the lowest level of program that people will write as it has one-to-one correspondence with Machine Code

Translation Programs - 3

- Interpreter
  - Translates HLL instructions to Lower Level instructions on-the-fly (at "run-time")
  - Generates equivalent Machine Code instructions for each HLL instruction
  - Examples: LISP, Prolog, BASIC, Java Bytecode
- Compilers and Assemblers work on whole files at a time and generate a separate executable version of the program, while Interpreter does not generate a separate Machine Code executable version of program
- Interpreted programs generally run much slower than machine code programs (compiled or assembled)

Summary

- PC and CPU are electronic devices
- CPU only understands 0s and 1s, i.e. Machine Code
- AL is low-level programming that has one-on-one correspondence with Machine Code
- AL is machine-specific (CPU-specific)
- AL is closest to the hardware (c.f. HLL) and allow efficient control of the hardware
- Some insight into PC and CPU hardware
- All programming languages require translation into Machine Code