EE 213 Computer Organization and Assembly Language

Week # 2, Lecture # 4

19th Dhu'l-Hijjah, 1439 A.H

3rd August 2018

These slides contains materials taken from various sources. I fully acknowledge all copyrights.

Minds open...

... Laptops closed

This presentation helps in delivering the lecture.

Take notes, interact and read text book to learn and gain knowledge.

Revision of Topics from Previous Lecture

- Cache
- Memory address range
- Hex to Binary
- Binary to Hex

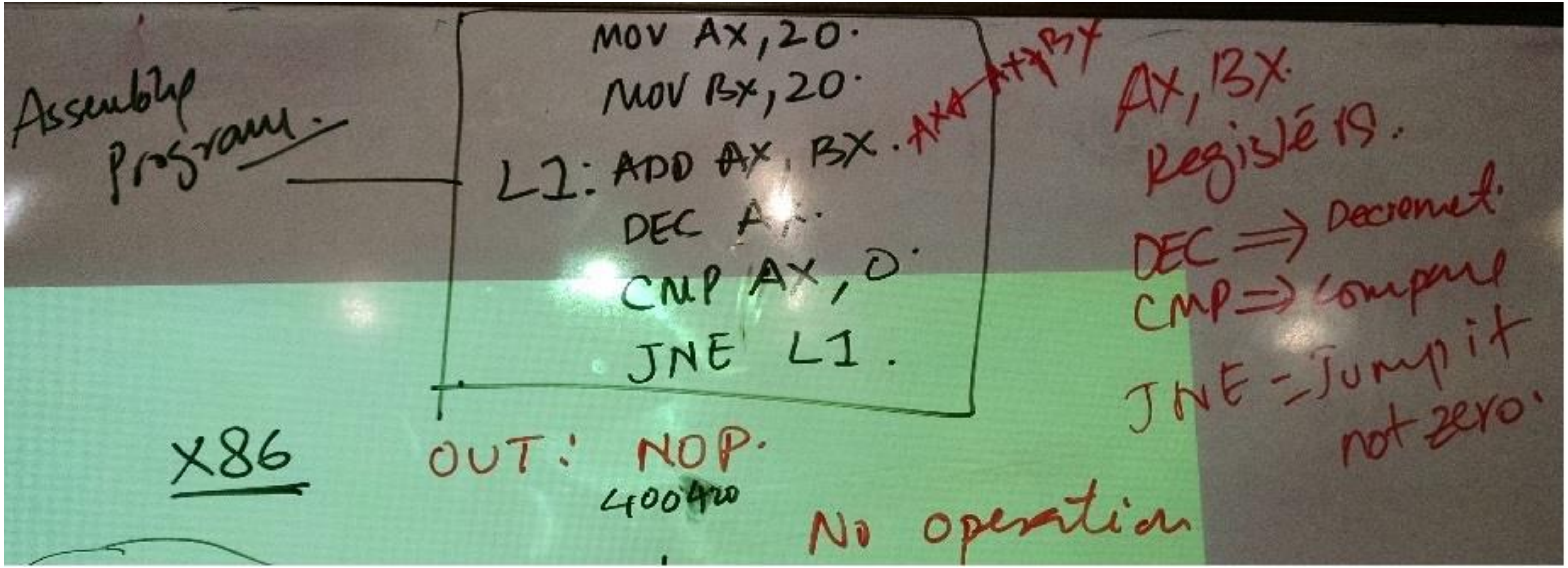
Learn in Lab. All labs contents are part of theory syllabus.

- Instruction Fetch and Execute
- Assembly Programs
  - High-level language are human friendly doesn’t shows hardware related details. Executable code contains one and zero which are difficult for humans.
  - Need a way to write programs that show processor details.
  - Assembly Language fills this gap by providing language statements which are closer to micro-architecture elements.
  - Therefore, the key goal of learning assembly is to understanding how HLL are executed on micro-architecture for better computational thinking.

Today’s Topics

- Understanding take-home assembly code
- What is machine code?
- Role of Compiler
- Role of Operating System
- Compilation, Linking and Loading of program for execution

Home work (Sec A)



Assembly program.

```
    MOV AX,20.
    MOV BX,20.
L1: ADD AX, BX.
    DEC AX.
    CMP AX,0.
    JNE L1.
```

AX ← AX+BX

AX, BX.
Registers.

DEC ⟹ Decrement.

CMP ⟹ Compare

JNE = Jump if not zero.

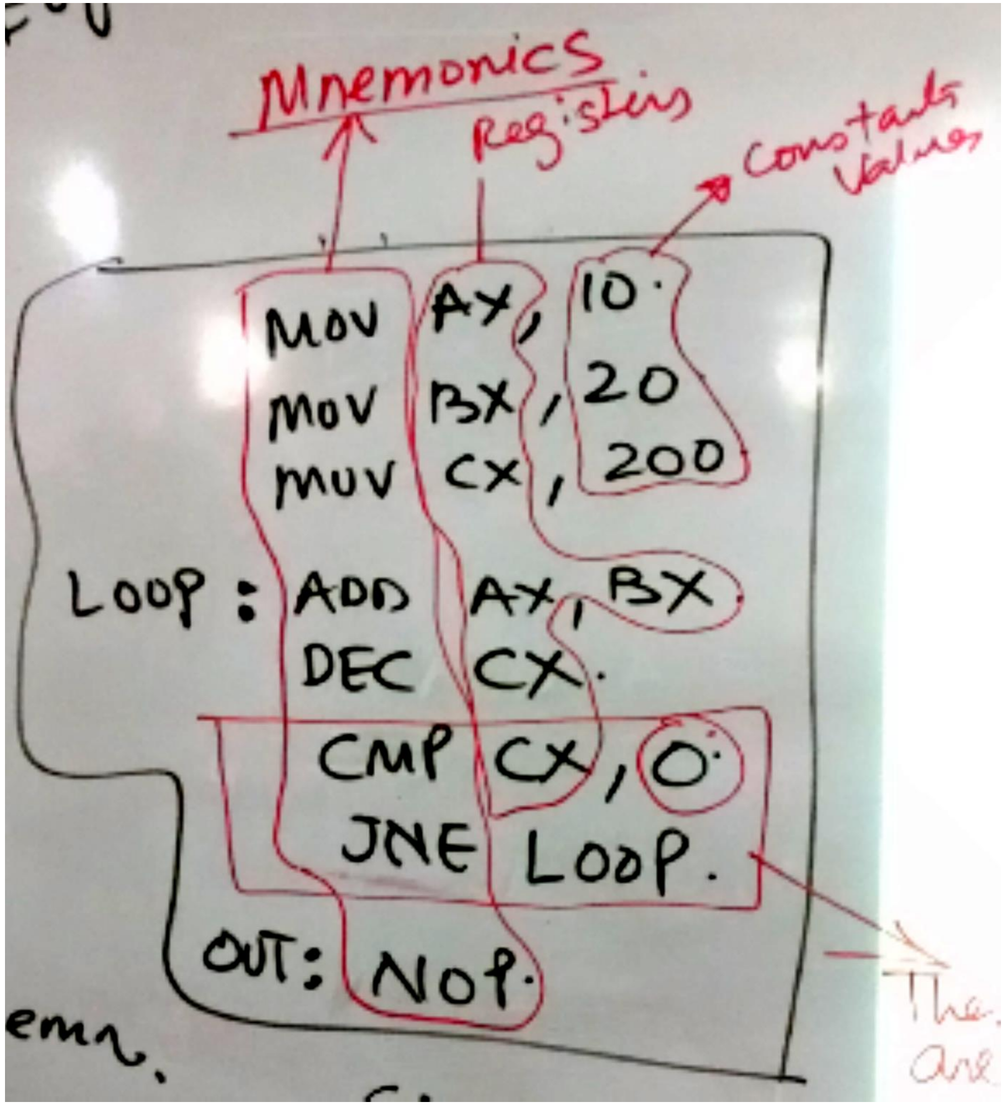
X86

OUT: NOP.

400420

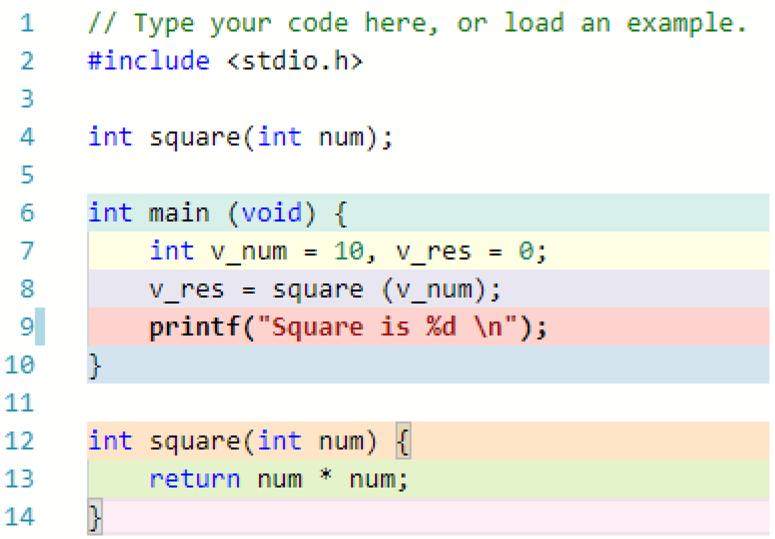
No operation

Home work (Sec E)

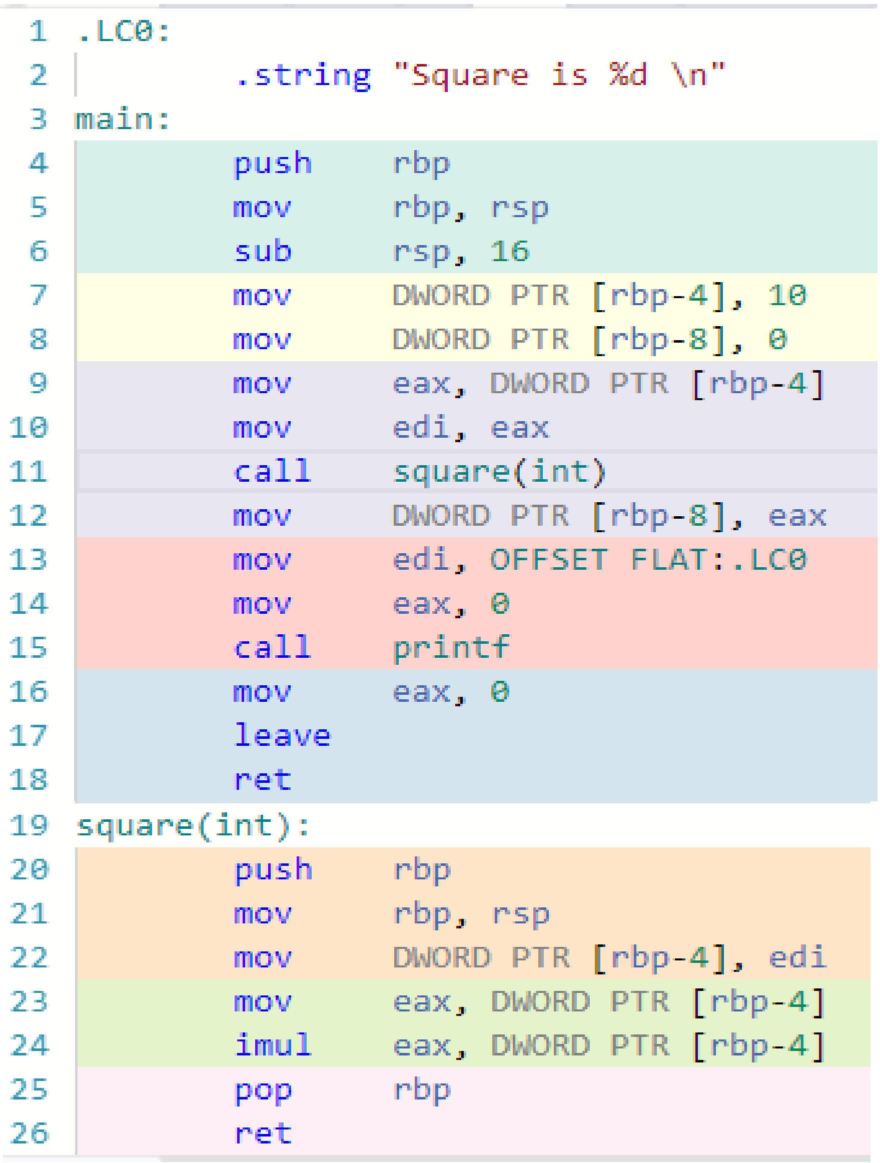


```
      MOV  AX, 10.
      MOV  BX, 20
      MUV  CX, 200

LOOP: ADD  AX, BX
      DEC  CX.
      CMP  CX, 0.
      JNE  LOOP.
 OUT: NOP.
```

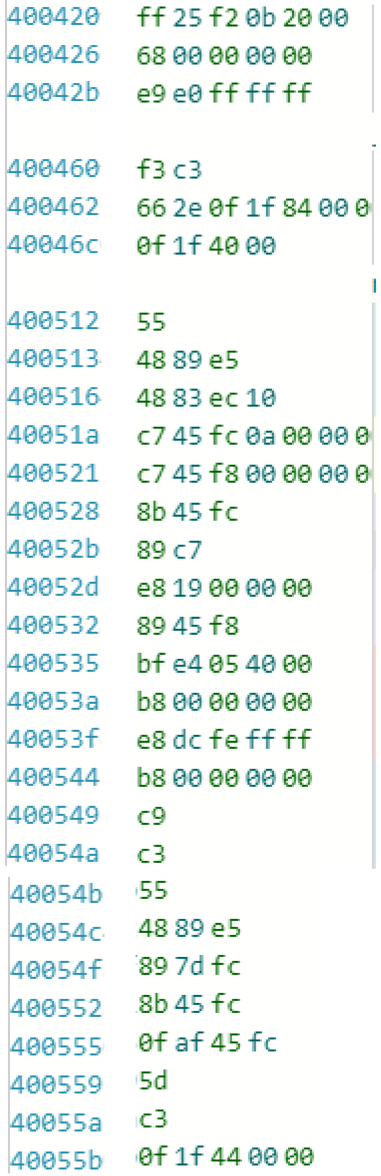
High-Level code <-> Assembly code <-> Machine code



```c
// Type your code here, or load an example.
#include <stdio.h>

int square(int num);

int main (void) {
    int v_num = 10, v_res = 0;
    v_res = square (v_num);
    printf("Square is %d \n");
}

int square(int num) {
    return num * num;
}
```



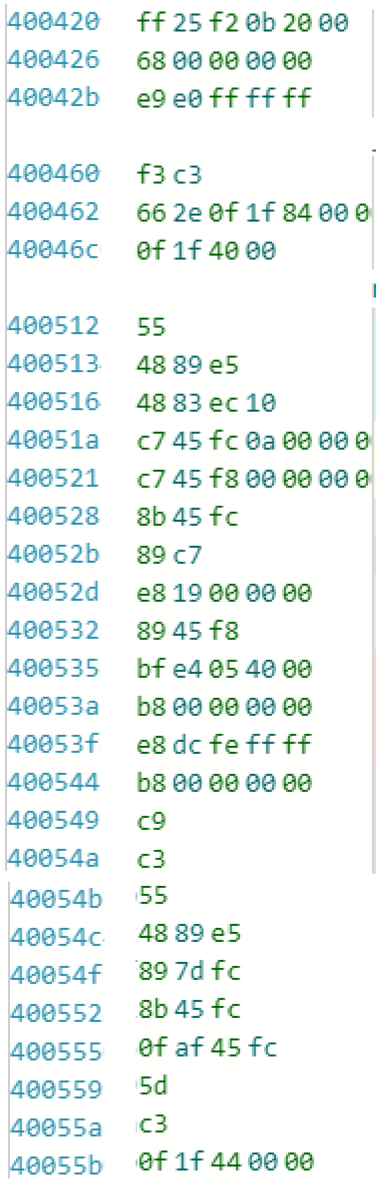
```asm
.LC0:
        .string "Square is %d \n"
main:
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     DWORD PTR [rbp-4], 10
        mov     DWORD PTR [rbp-8], 0
        mov     eax, DWORD PTR [rbp-4]
        mov     edi, eax
        call    square(int)
        mov     DWORD PTR [rbp-8], eax
        mov     edi, OFFSET FLAT:.LC0
        mov     eax, 0
        call    printf
        mov     eax, 0
        leave
        ret
square(int):
        push    rbp
        mov     rbp, rsp
        mov     DWORD PTR [rbp-4], edi
        mov     eax, DWORD PTR [rbp-4]
        imul    eax, DWORD PTR [rbp-4]
        pop     rbp
        ret
```



```
400420   ff 25 f2 0b 20 00
400426   68 00 00 00 00
40042b   e9 e0 ff ff ff

400460   f3 c3
400462   66 2e 0f 1f 84 00 0
40046c   0f 1f 40 00

400512   55
400513   48 89 e5
400516   48 83 ec 10
40051a   c7 45 fc 0a 00 00 0
400521   c7 45 f8 00 00 00 0
400528   8b 45 fc
40052b   89 c7
40052d   e8 19 00 00 00
400532   89 45 f8
400535   bf e4 05 40 00
40053a   b8 00 00 00 00
40053f   e8 dc fe ff ff
400544   b8 00 00 00 00
400549   c9
40054a   c3
40054b   55
40054c   48 89 e5
40054f   89 7d fc
400552   8b 45 fc
400555   0f af 45 fc
400559   5d
40055a   c3
40055b   0f 1f 44 00 00
```

What is machine code?

- Machine code is a computer program written in machine language instructions that can be executed directly by a processor.
- Machine code is strictly numerical and may be regarded as the lowest-level representation of a program or as a hardware-dependent programming language.
- It is possible to write programs directly in machine code, but it is tedious and error prone to manage individual bits and calculate numerical addresses and constants manually.
- Programs are very rarely written directly in machine code in modern contexts. Machine coding is done for low level debugging, program patching, etc.



```
400420   ff 25 f2 0b 20 00
400426   68 00 00 00 00
40042b   e9 e0 ff ff ff

400460   f3 c3
400462   66 2e 0f 1f 84 00 0
40046c   0f 1f 40 00

400512   55
400513   48 89 e5
400516   48 83 ec 10
40051a   c7 45 fc 0a 00 00 0
400521   c7 45 f8 00 00 00 0
400528   8b 45 fc
40052b   89 c7
40052d   e8 19 00 00 00
400532   89 45 f8
400535   bf e4 05 40 00
40053a   b8 00 00 00 00
40053f   e8 dc fe ff ff
400544   b8 00 00 00 00
400549   c9
40054a   c3
40054b   55
40054c   48 89 e5
40054f   89 7d fc
400552   8b 45 fc
400555   0f af 45 fc
400559   5d
40055a   c3
40055b   0f 1f 44 00 00
```

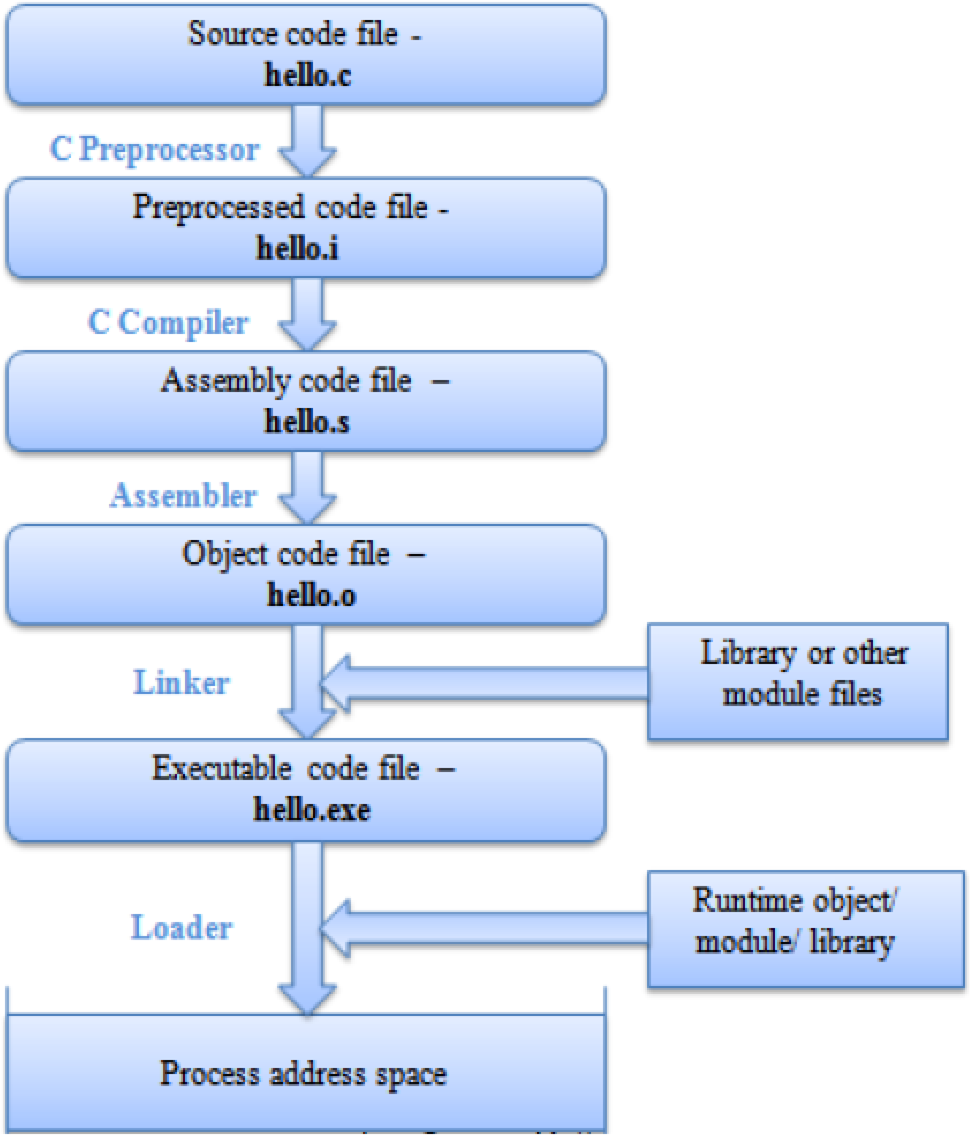
Role of Compiler

- Compiler converts high-level code into machine code (stored in .exe file) which will be executed by the processor (a complex digital circuit).
- There could be many different ways to design digital circuits. How compiler knows about the processor?
- So, there is a unique compiler for each processor. (Why?)
- Compiler read each high-level language statement and break the computation in each statement in terms of operations on data. For example, c = a + b means that there are three variable (memory locations) a, b, c and contents of a and b are added together and stored in c.
- Therefore, compiler generated code is for a specific processor. The code contains hundred of operations in specific order. The operations are in binary and act as a instruction to the processor.
- Therefore, the processor is suppose to read each instruction and execute it step-by-step and store the results internally or in memory.

Role of Operating System

- Compiler makes executable file on disk.
- Operating System (OS) reads code from disk and load it into memory.
- OS later create a process to execute the program on the processor.
- Processor (e.g. Intel Core i7, AMD, IBM, NVIDIA) executes code in memory by reading inputs: keyboard or data files on disk, etc. and generating outputs: Display, Ports (network, printer, etc.), disk, or other connected devices.
- Therefore, OS give users a user-friendly computing environment where multiple programs execute together facilitating the computer user.
- However, in this course, we are interested in understanding:
  - (40%) How internal digital circuits of a processor are organized to execute machine-code? (No circuit diagrams only block diagram of processor organization)
  - (60%) How processor perform execution steps using the internal organization when it executes each machine code instruction?

Compiling HLL programs into Machine Code



Source code file - **hello.c**

C Preprocessor

Preprocessed code file - **hello.i**

C Compiler

Assembly code file – **hello.s**

Assembler

Object code file – **hello.o**

Linker

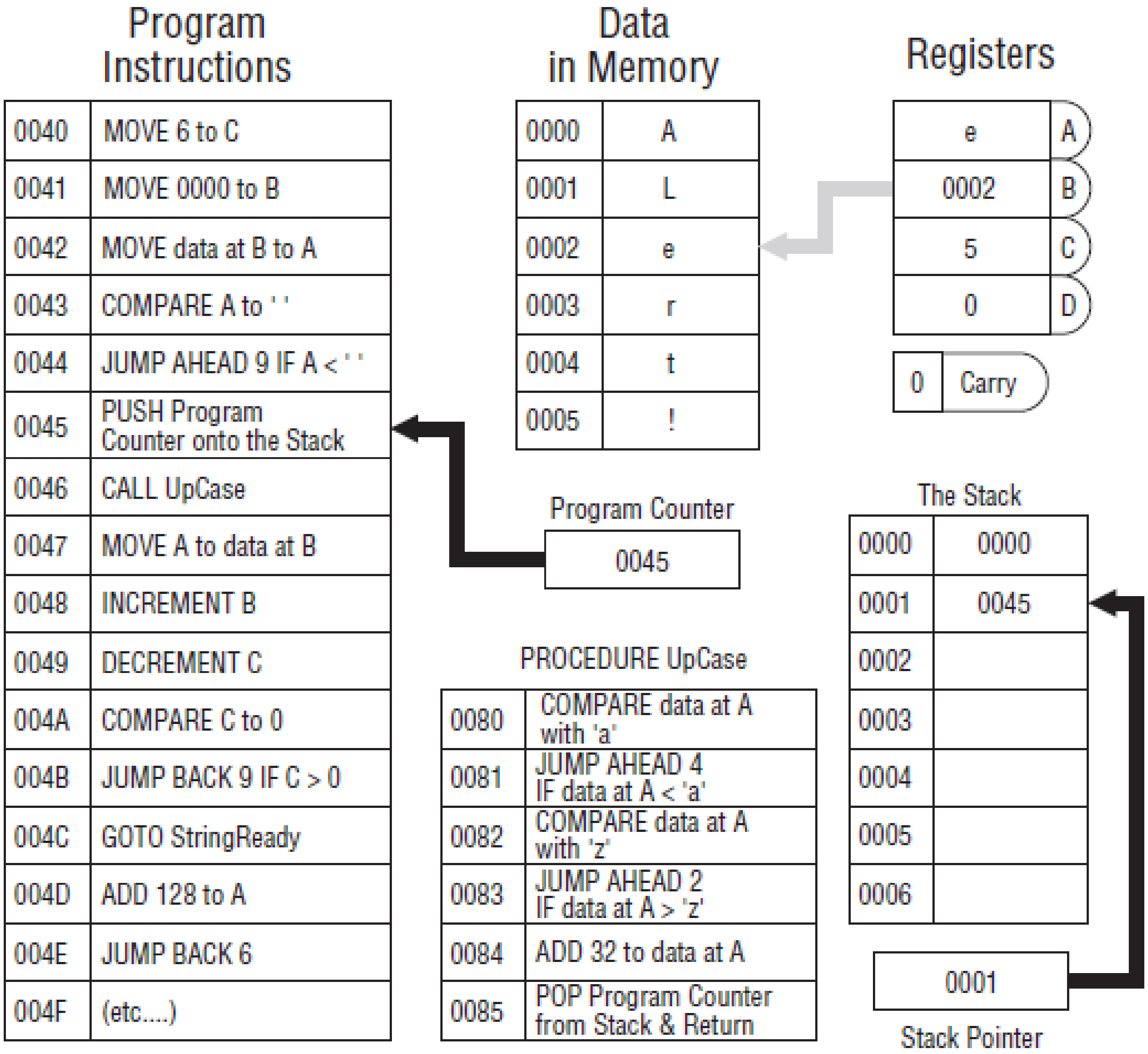
Library or other module files

Executable code file – **hello.exe**

Loader

Runtime object/ module/ library

Process address space



Program Instructions

0040	MOVE 6 to C
0041	MOVE 0000 to B
0042	MOVE data at B to A
0043	COMPARE A to ' '
0044	JUMP AHEAD 9 IF A < ' '
0045	PUSH Program Counter onto the Stack
0046	CALL UpCase
0047	MOVE A to data at B
0048	INCREMENT B
0049	DECREMENT C
004A	COMPARE C to 0
004B	JUMP BACK 9 IF C > 0
004C	GOTO StringReady
004D	ADD 128 to A
004E	JUMP BACK 6
004F	(etc....)

Data in Memory

0000	A
0001	L
0002	e
0003	r
0004	t
0005	!

Registers

e	A
0002	B
5	C
0	D

0	Carry

Program Counter

0045

PROCEDURE UpCase

0080	COMPARE data at A with 'a'
0081	JUMP AHEAD 4 IF data at A < 'a'
0082	COMPARE data at A with 'z'
0083	JUMP AHEAD 2 IF data at A > 'z'
0084	ADD 32 to data at A
0085	POP Program Counter from Stack & Return

The Stack

0000	0000
0001	0045
0002	
0003	
0004	
0005	
0006	

0001

Stack Pointer

Towards understanding Micro-architecture

- We want to understanding (in block diagram form) the **micro-architecture** of a processor which implements the **ISA**.
- **Micro-architecture** of a processor:
  - Different vendors may design different micro-architectures (Why?)
  - Micro-architecture is very complex sequential circuit which is programmable i.e. it fetches instructions, decode them, fetches data (from processor registers or memory) and execute (i.e. apply operation specified in the instruction on data operands).
  - Instructions given to a micro-architecture obeys the Instruction Set Architecture (ISA) which is unique to each micro-architecture.
  - Compilers need to know about processor's micro-architecture elements and ISA in order to convert a high-level language code to machine code.
- How micro-architecture is designed?
- How executable code generated by compiler get executed on the micro-architecture?