CS220 – Logic Design
AS01-What is Assembly?

• Outline
  – Processor Instructions
  – High-Level Languages
  – Assembly Language
To the microprocessor (CPU) a program is a sequence of binary codes stored in memory.

The CPU can perform a certain set of operations and for each operation there is a corresponding binary instruction code.

One code might move data from memory to a register, another might add data in two registers, another might multiply values in registry and in memory, etc.
AS01-What is Assembly? Processor Instructions

• The instruction code is typically stored in one area of memory, while data the program uses is in another.

• The program **counter** is a processor register that contains the address of the next instruction code to be executed.

• Local data is stored in a memory area known as the stack. The stack pointer always contains the address of the last item stored.
AS01-What is Assembly?
Processor Instructions

The stack pointer is decremented as data is added to the stack.

Successive instructions are read from increasingly higher addresses.
Intel IA-32 processors use varying length instruction codes. Each code consists of four parts: (1) Optional instruction prefix, (2) the opcode (operational code), (3) optional modifier and, (4) optional data element.
AS01-What is Assembly?  
Processor Instructions

- The opcode is the only **required** part of the instruction code. Each operation the processor can perform (move, add, multiply, compare, branch, etc) has a unique opcode.

- The optional instruction prefixes are arranged in four groups: (1) lock and repeat prefixes, (2) **segment** override and branch hint prefixes, (3) operand size override prefix and, (4) address size override prefixes.
AS01-What is Assembly?

Processor Instructions

- The ModR/M modifier is used to indicate the addressing mode and to indicate which processor registers are used by the opcode.
- The SIB (scale-index-base) modifier defines the registers used in array operations.
- The address displacement modifier is used to define an offset to a memory location that is used by the opcode.
AS01-What is Assembly?
Processor Instructions

- The final part of the instruction code is a static data value. (Most instructions get data values from memory or processor registers.)

- An example IA-32 instruction is:

  C7 45 FC 01 00 00 00

  The opcode is C7 (move value to mem), 45 FC defines a memory location (-4 bytes (FC) from the address in the EBP register (45)). The data value is 1 (or actually 00000001).
Writing programs directly in binary instruction code (also known as machine code) would be extremely tedious.

In assembly language programming, easy to remember instruction mnemonics are used instead of binary opcodes. Registers are referred to by name instead of by binary code. Special notation is used to represent the various addressing modes.
## AS01-What is Assembly?

### Opcode Mnemonics

<table>
<thead>
<tr>
<th>Machine Code</th>
<th>Assembly Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>55</td>
<td><code>push %ebp</code></td>
</tr>
<tr>
<td>89E5</td>
<td><code>mov %esp, %ebp</code></td>
</tr>
<tr>
<td>83EC08</td>
<td><code>sub $0x8, %esp</code></td>
</tr>
<tr>
<td>C745FC01000000</td>
<td><code>movl $0x1, -4(%ebp)</code></td>
</tr>
<tr>
<td>83EC0C</td>
<td><code>sub $0xc, %esp</code></td>
</tr>
<tr>
<td>6A00</td>
<td><code>push $0x0</code></td>
</tr>
<tr>
<td>E8EFCE7A00</td>
<td><code>call 8048348</code></td>
</tr>
</tbody>
</table>
AS01-What is Assembly?  
Assembly Programs

- Standard IA-32 assembly language mnemonics include: `mov`, `add`, `sub`, `mul`, `div`, `cmp`, `jmp`, `push`, `pop`.
- There is a one-to-one correspondence between an instruction mnemonic and an instruction opcode.
- In addition to opcode mnemonics assembly programs can also contain data definitions and assembler directives.
AS01-What is Assembly?
Data Definitions

- A global data definition consists of a label, data type and value. The label also has a value (equal to the address of the data):
  - `testvalue: .long 150`
  - `message: .ascii "This is a test message"
  - `pi: .float 3.14159`

- The data is stored in consecutive memory locations as shown on the following slide. (Global data is not stored on the stack!!!)
The bytes are shown in hex notation. **testvalue** is stored in 4 bytes. **message** is stored in ASCII. **pi** is stored in IEEE single precision format (in 4 bytes).

Note that testvalue is stored in “little-endian” order.
AS01-What is Assembly?

Data Definitions

Here is an snippet of assembly code that copies the value at address `testvalue` to the EBX register, adds 10 to the register and writes the value back to address `testvalue`:

```
movl  testvalue, %ebx
addl  $10, %ebx
movl  %ebx, testvalue
```
Data can also be stored on the stack. The stack is used for local variable storage and for passing arguments to a function during a function call.

The **push** and **pop** mnemonics are used to store and retrieve data from the stack. The stack pointer register (**ESP**) contains the address of the last item that was pushed onto the stack.
Assembler Directives

• Assembler directives are used to tell the assembler how to perform special functions.

• The `.long`, `.ascii`, and `.float` words that were used earlier (when defining data) are examples of assembler directives.

• The `.section` directive is used to define the text, data, and bss (uninitialized data) sections that are contained in all programs.
Note: C++ external and static local variables are stored in the data or bss sections. **Automatic** locals are stored on the stack:

```c
// data:   bint, dint, hint
// bss:    aint, cint, gint
// stack:  eint, fint
int   aint, bint = 100;
static int cint, dint = 200
void foo()
{ int eint, fint = 300;
  static int gint, hint = 400; }
```
AS01-What is Assembly?

Portability

- Different processors use different binary opcodes for similar operations.
- Because of differences in architecture (number and types of registers) different processors have different capabilities and correspondingly different assembly language mnemonics and register names.
- Both binary machine code and assembly language are non-portable.
AS01-What is Assembly?
High-Level Languages

- The source code for high-level languages (C++, Lisp, Java) is typically easy to port from one type of processor to another. (Ideally, no changes to the source code would be required to run the code on a different processor.)

- There are three types of high-level languages (HLLs): compiled, interpreted, and hybrid.
C, C++, and Fortran are examples of compiled languages.

Creating an executable program requires two steps: (1) source code files are compiled into machine code stored in object code files, (2) object files are linked to create the program.

One statement in an HLL will usually compile to multiple machine language opcodes.
AS01-What is Assembly?
Compiled Languages
AS01-What is Assembly?
Interpreted Languages

- Scheme, Perl, Python, Windows batch files, and UNIX shell scripts are examples of interpreted languages.
- An interpreter program converts the source code into the proper processor instructions. The source code is the “program”.
- Interpreted programs run more slowly than compiled programs, but programs can be debugged quickly.
Java and C# are examples of hybrid languages.

Source code is compiled into “byte code” that runs in a “virtual machine”. The byte code can run on any platform that supports the virtual machine.

Code runs nearly as fast as compiled code but has the advantage of being portable.
AS01-What is Assembly? Why Learn Assembly?

- You will gain a much better understanding of how processors and computers work.
- You will learn a lot more about how HLLs (like C++) really work.
- You may have to use assembly to access special CPU features.
- Assembly code may occasionally be faster and smaller than compiled code.