Lecture 13 – Real world architectures

NB: lecture notes define expected knowledge in this section.

A stack is a data structure for accessing data in LIFO (last in, first out) order.

Data items can be pushed onto a stack, and popped off

e.g. (pseudocode)

```
push (3)
push (-5)
push (8)
x = pop ()
push (4)
y = pop ()
z = pop ()
```

→ results in $x = 8$

$y = 4$

$z = -5$.

~~On modern process~~

In modern computers, every running ~~process~~ program (actually, every thread

has a *call stack*, also known as "the stack".

Details vary, but typical usage is that on every method call, ~~the~~ f

- parameters
- return address
- local variables

are pushed onto the stack. This collection of data is the method's *frame* or *stack frame*.

When the method returns, these are all popped off the stack and execution continues from the popped return address.

(This is better than MARIE's way of calling subroutines, because it permits recursion.)

The stack of methods you see in a debugger is derived from the call stack.

demo? - yes - StackDemo.java

Intel's 32-bit architecture ("IA32") is an important real-world architecture. (Basis for 80386, x86, Pentium, Core 2 Duo, e

Include:

- 4 general-purpose registers, EAX, EBX, ECX, EDX
- a program counter or <u>instruction pointer</u>, EIP
- a stack pointer ESP (points at top of stack)
- a base pointer EBP (points to base of current method's stack frame).
- various registers for memory management, called <u>segment registers</u> because memory is divided into <u>segments</u>.
- a <u>status register</u> (EFLAGS) that stores the status of various operations.

~~demo - ~~assem~~ compile simple.c~~
~~simple2.c~~
~~into assembly & show.~~

IA32 includes many useful instructions, e.g.

- immediate, direct & indirect add, load, store, multiply, ~~etc~~ jump
  (details next lecture

- a CALL instruction for invoking methods

- a LOOP instruction that decrements a counter & jumps to top of a loop

- atomic increment & decrement instructions

- many instructions for operating systems functionality
  e.g. interrupts, switching between processes, memory management, security & protection.

demo: compile simple.c
simple2.c
etc. look at assembly language.

exercise: ① produce assembly `gcc -S -o simple.S simple.c`

② edit in text editor to change functionality
e.g. - add a third number.
- loop to a different end value.

```
gcc -o simple.exe simple.c
```

MIPS is another important architecture (used in, e.g. TIVO, Playstation I & II).

- has 32 general purpose registers
- is a load/store architecture, meaning every instruction (except load and store) use registers as operands. (ie. no memory locations as operands, except in LOAD & STORE).