Lecture 27

Stack

- a region of memory for temporary storage of data
- Last in, first out
- stack starts from larger memory addresses then decreases
- stack pointer: `sp` register points to the top of the stack ### Example Stack before and after adding 2 words -> `sp` is decreased by 8
- Saving and restoring registers
  - Functions should not trample registers of the caller function
- Preserved Registers
  - must contain some registers at beginning and end of called function
  - caller expects them to be the same after the call
- Nonpreserved registers
  - do not need to be preserved
- Calling conventions are not enforced by hardware

```
_start: addi s1, zero, 77
    addi s2, zero, -1
    addi s3, zero, 0
    addi a0, zero, 11
    # same as last lecture
    addi a5, zero, 66
    jal add6
    sub s4, s1, s2
    sub s6, s3, s1
END: ebreak
add6:   addi sp, sp, -12
    sw s1, 8(sp)
    sw s2, 4(sp)
    sw s3, 0(sp)
    add s1, s0, s1
    add s2, a2, a3
    add s3, a4, a5
    add s1, s1, s2
    add a0, s1, s3
    lw s3, 0(sp)
    lw s2, 4(sp)
    lw s1, 8(sp)
    addi sp, sp, 12
    jr ra
```

- Nested subroutines

```
int main() {
    return add6(11,22,33,44,55,66);
}
```

```
int add6(int a, int b, int c, int d, int e, int f) {
    return add3(a,b,c) + add3(d,e,f);
}
int add3(int x, y, z) {
    return x + y + z;
}
```

- In assembly,

```
# main same as before
add6:   addi sp, sp, -4 # allocate space on stack
    sw ra, 0(sp) # save ra
    jal add3 # overwrites ra to be ret1
ret1:   add t0, zero, a0
    add a0, zero, a3
    add a1, zero, a4
    add a2, zero, a5
    jal add3
    add a0, t0, a0
    lw ra, 0(sp) # restore ra
    addi sp, sp, 4 # deallocate stack
add3:   add a0, a0, a1
    add a0, a0, a2
    jr ra
```

- caller save: t0-t6, a0-a7 (stacked calls)
- callee save: s0-s11, ra (restore before return)