Computer Systems Organisation

Summer 2021, IIIT Hyderabad

08 June, Tuesday (Tutorial 2)

Taught by Aakash {Aanegola, Jain}

Assembly

Consider the C code

```
int main ()
{
    int a = 1;
    int b = 3;
    int c = 1+3;
    printf("%d\n",c);
    return 0;
}
```

The equivalent assembly is:

```
push %rbp
mov  %rsp, %rbp
sub  $0x20, %rsp
```

These three lines are stack manipulation as part of calling `main()`.

```
movq $0x1, -0x18(%rbp)
move $0x3, -0x10(%rbp)
```

These lines assign values to `a` and `b` in *memory locations*.

```
mov  -0x18(%rbp), %rdx
mov  -0x10(%rbp), %rax
```

`a` and `b` are moved to registers preparatory to carrying out the addition.

```
add  %rdx, %rax
mov  %rax, -0x8(%rbp)
```

The value of `c` is computed and moved to memory.

```
mov  -0x8(%rbp), %rax
```

The value of `c` is moved back to a register for the `printf()` call.

Stack

When a function is called, the value of `%rbp` is pushed and then `%rbp` is changed to point to `%rsp`. Now, since the base of the stack is at `%rsp`, none of the frames below it are "visible" to the current function. When the function is exited, the

old `%rbp` value is popped and moved back to the register, thereby making the rest of the stack visible.