COMP1917: Computing 1

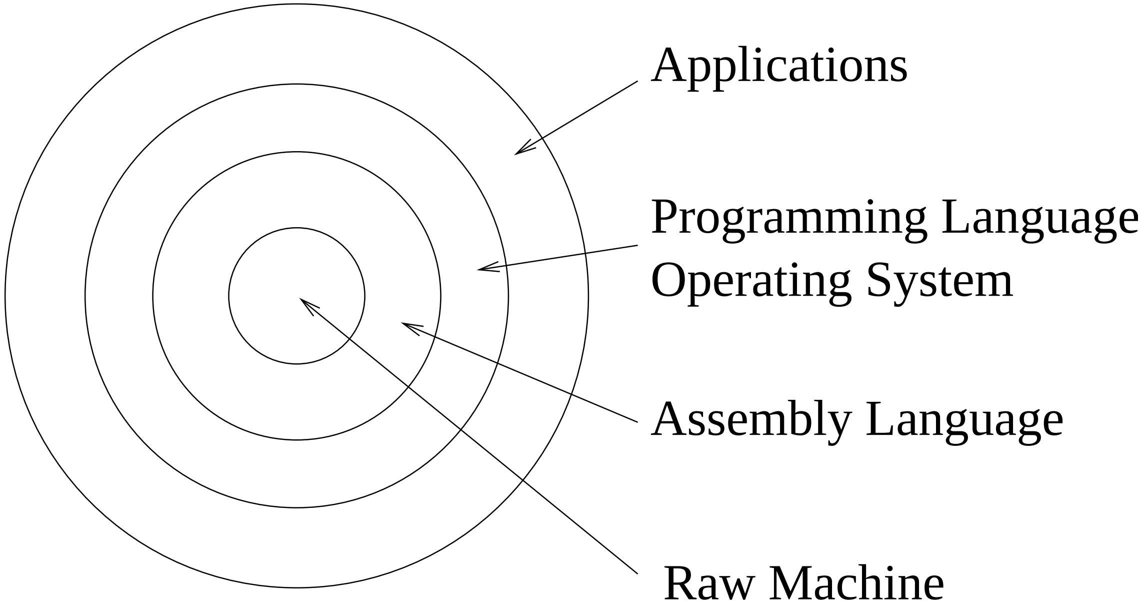
17. Memory and Stack Frames

Overview

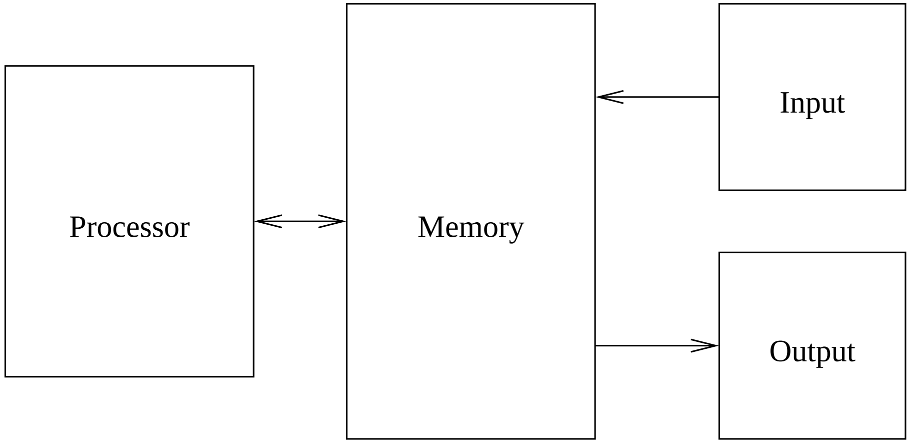
- Computer Systems
- Memory Map
- Static and Dynamic Variables
- Function Calls
- Stack Frames
- Stack Overflow

Computer Systems

Modern computer systems are layered.

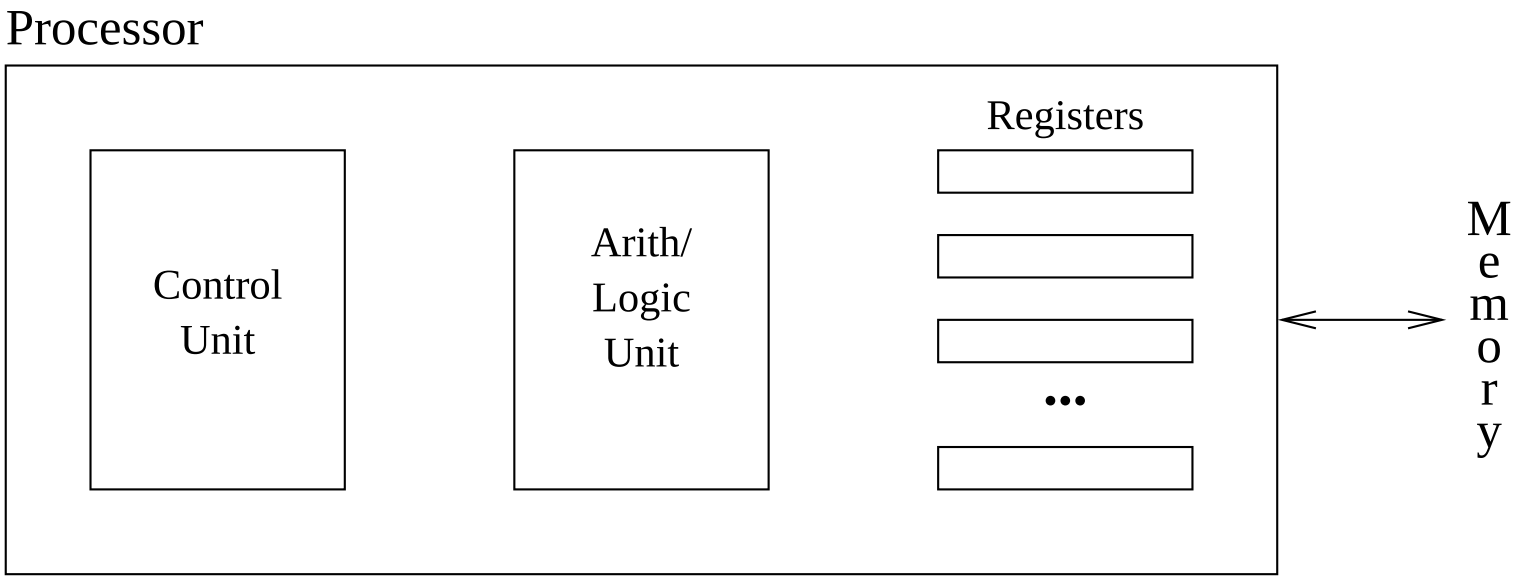


Computer Architecture



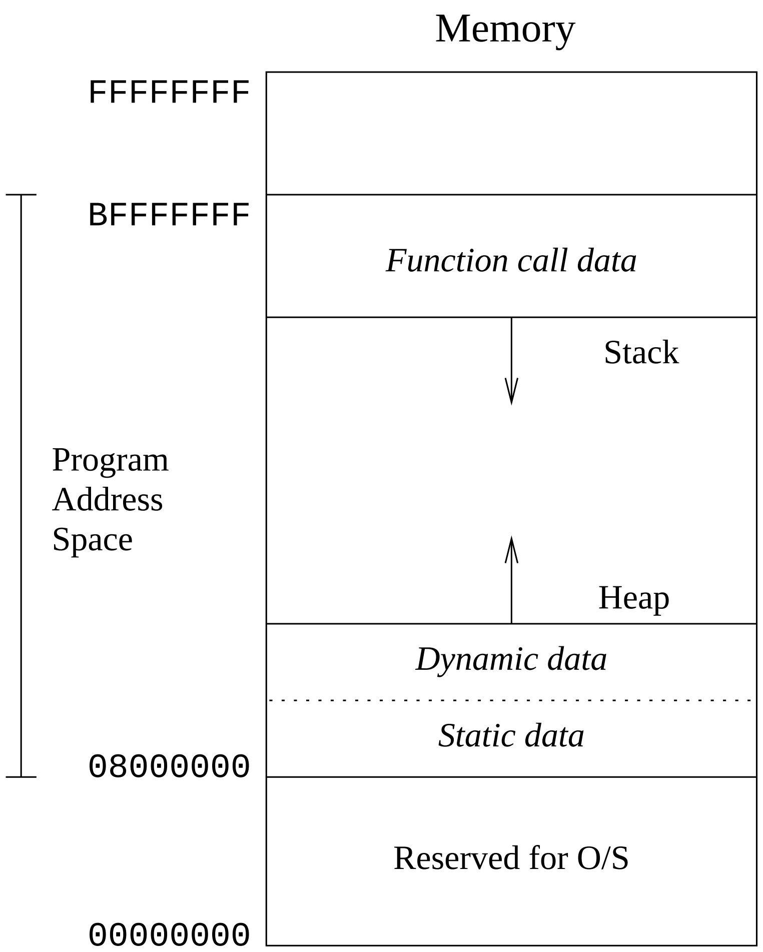
- Processor: control, calculation
- Memory: data & program storage
- Input/output: interface to the world

Central Processing Unit



Registers are used as “working memory” to store intermediate values in a computation.

Memory Map


Memory
FFFFFFFF
BFFFFFFF
Function call data
Stack
Program
Address
Space
Heap
Dynamic data
Static data
08000000
Reserved for O/S
00000000

Static Variables

Recall: static variables keep their value from one function call to the next.

What is the output of this code?

```
void inc()
{
   static int k = 5;
   int l = 5;

   k++;
   l++;
   printf("k = %d, l = %d\n", k, l );
}
```

```
int main( void )
{
    inc();
    inc();
}
```

Static and Dynamic Variables

```
{
  static int k;
  int  l;
  int *a =(int *)malloc( 10 * sizeof( int ));

  printf(" static variable k  is stored at  %8X\n", &k );
  printf("dynamic variable a  is stored at  %8X\n",  a );
  printf(" local  variable l  is stored at  %8X\n", &l );
}
```

Output:

```
 static variable k  is stored at   80496F0
dynamic variable a  is stored at   804A008
 local  variable l  is stored at  BFD710D0
```

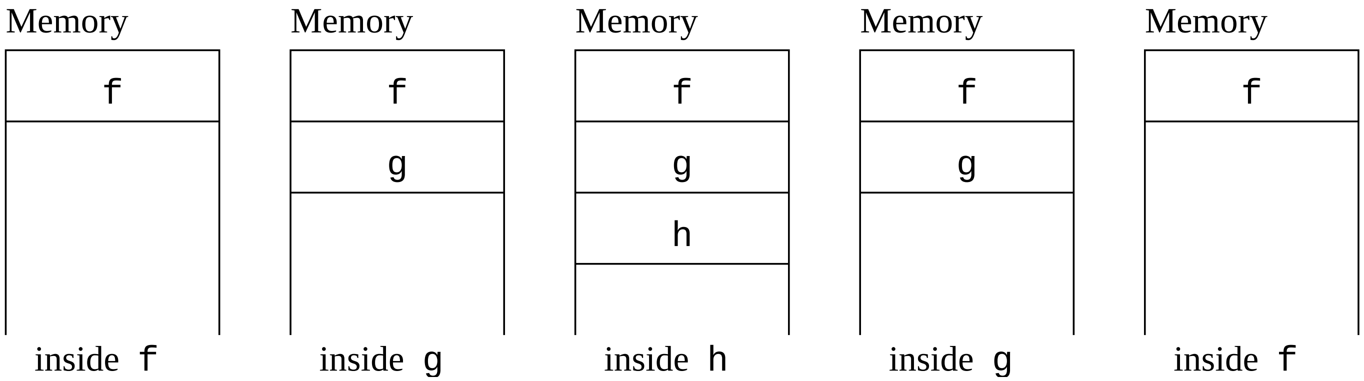
Function Calls

If `main` calls `f` calls `g` calls `h` ...

Then `h` finishes, then `g` finishes, then `f` finishes and we're back in `main`.

Function call/return is last-in, first-out (LIFO) protocol.

⇒ use a stack of return addresses.



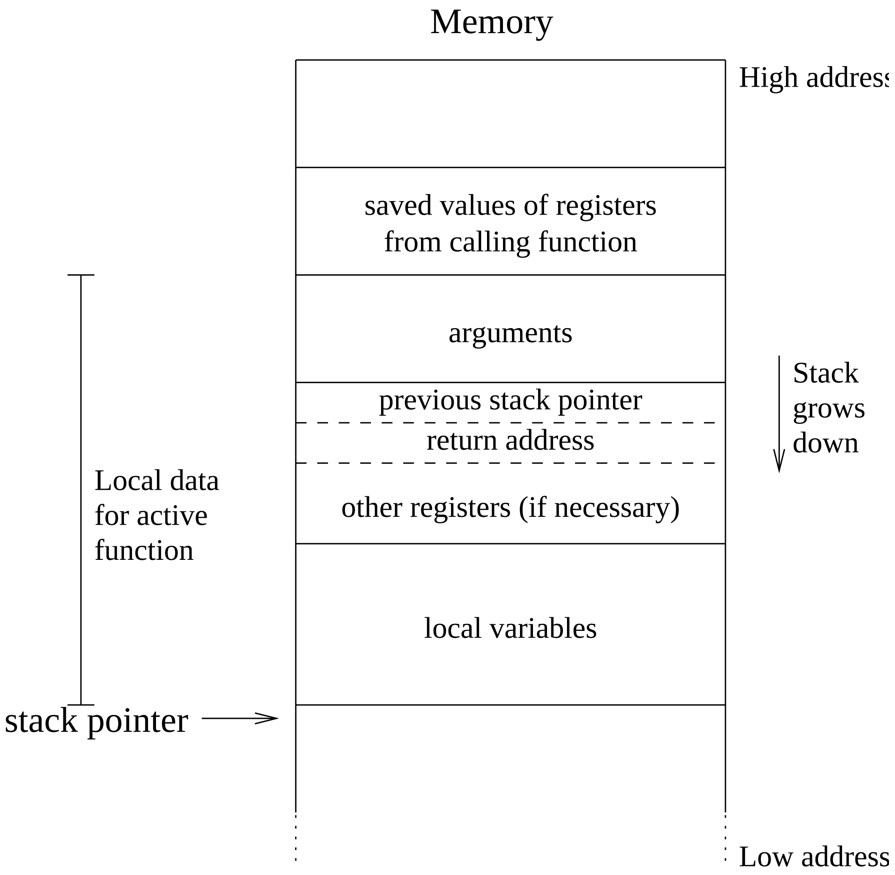
Stack Frames

- Inside a function, we need access to:
  - arguments
  - local variables
- When the function terminates, need to retrieve
  - return value
  - register values from previous function
  - previous stack pointer
  - return address

All of these are located on the stack. Thus, a small region on the stack is associated with the invocation of each function.

This is called a stack frame.

gcc Stack Frame



Creating a Stack Frame

On entry to a function:

1. compute size of stack frame and new stack pointer
2. allocate memory for frame
3. save registers
4. store arguments
5. save previous `stack pointer` and `return address`
6. change `stack pointer`
7. pass control to new function

Removing a Stack Frame

On exit from a function:

1. save the return value
2. restore previous register values
3. pop stack frame by reverting to previous `stack pointer`
4. restore control to previous function by jumping to `return address`

factorial.c

```
int factorial( int n )
{
   printf("n at %X is equal to %d\n", &n, n );
   if( n <= 1 )
       return( 1 );
   else
       return( n * factorial( n-1 ));
}
int main( void )
{
   int fact; int n;
   printf("Enter number: ");
   scanf( "%d", &n );
   fact = factorial( n );
   printf("Factorial of %d is %d\n", n, fact );
}
```

Output of `factorial.c`

```
Enter number: 4
n at BFD8E1B0 is equal to 4
n at BFD8E190 is equal to 3
n at BFD8E170 is equal to 2
n at BFD8E150 is equal to 1
Factorial of 4 is 24
```

Stack Overflow

```
Enter number: 1000000
n at BFD8E1B0 is equal to 1000000
n at BFD8E190 is equal to 999999
n at BFD8E170 is equal to 999998
n at BFD8E150 is equal to 999997

...

n at BF46E700 is equal to 738089
n at BF46E6E0 is equal to 738088
n at BF46E6C0 is equal to 738087
Segmentation fault
```