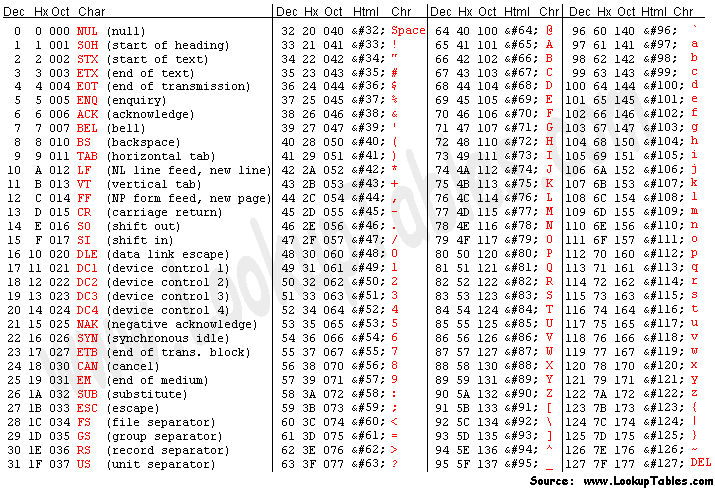
* From **H to D**:
  + Take the farthest number to the right and multiply it like so:
    - Ex: 0xDF23

(3\*16^0)+(2\*16^1)+(16\*16^2)+(13\*16^3)

* + - to get 57123
* From **D to H**
  + Divde my 16 and take the remainders and reverse the order
    - Ex: 23834
    - 23834/16 = 1484 R 10 (A)
    - 1484/16 = 93 R 1 (1)
    - 93/16 = 5 R 13 (D)
    - 5/16 = 0 R 5 (5)
    - now read bottom to top for the remainders
    - to get 0x5D1A
* **Transposition ciphers**:
  + They change the order of the characters, but no the characters themselves
  + Columnar transposition:
    - Takes a base word as the top of the column and one you figure out the base word all the other letters should line up
    - You generally do not include spaces
  + Scytale cipher:

| | | | | | |

| H | E | L | P | M | |

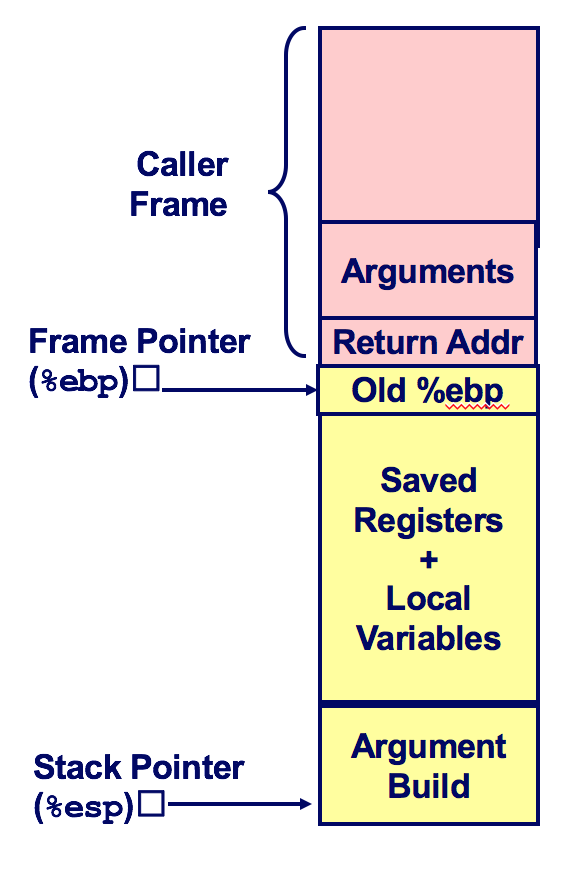
\_\_| E | I | A | M | U |\_\_|

| | N | D | E | R | A |

| | T | T | A | C | K |

| | | | | | |

* **Monoalphabetic** ciphers keep ordering od the characters the same, but implement a fixed substitution of plaintext characters to cipher text characters:
  + Caesar Cipher:
    - Each letter of plaintext is replaced by a letter a fixed number of positions down the alphabet
    - Often implemented with a wheel
* **Polyalphabetic** ciphers are substitution ciphers that use multiple alphabets to encrypt messages
  + Vigenere Cipher:
    - Use multiple Caesar ciphers on plaintext
* **Public Key Encryption**:
  + Key distribution problem simplified, there are not shared secret keys unlike the enigma
  + Relies upon a difficult math problem to protect you encrypted data
  + There are two parts to public key encryption, there is the public key that is published by the recipient and there is the private key that is kept secret by the recipient
  + Sender encrypts data using the public key
  + Recipient decrypts data using private key
* Dominating Set encryption:



08048394 <f>:

8048394: push %ebp

8048395: mov %esp,%ebp

8048397: sub $0x10,%esp

804839a: movl $0x0,-0x8(%ebp)

80483a1: movl $0x1,-0x4(%ebp)

80483a8: addl $0xb,-0x8(%ebp)

80483ac: mov -0x4(%ebp),%eax

80483af: sub %eax,-0x8(%ebp)

80483b2: subl $0x1,-0x8(%ebp)

80483b6: addl $0x1,-0x4(%ebp)

80483ba: leave

80483bb: ret

**Arithmetic Operators**

void f(){

int a = 0;

int b = 1;

a = a + 11;

a = a - b;

a--;

b++;

}

int main() { f();}

|  |
| --- |
| **Global Variables**  **ELF header**  nMagic number, type (.o, exec, .so), machine, byte ordering, etc.  Program header table  nPage size, virtual addresses of memory segments (sections), segment sizes, entry point  .text section  nCode  .data section  nInitialized (static) data  .bss section  nUninitialized (static) data  “Block Started by Symbol”  .symtab section  nSymbol table  nProcedure and static variable names  nSection names and locations  .rel.text section  nRelocation info for .text section  nAddresses of instructions that will need to be modified in the executable  nInstructions for modifying.  .rel.data section  nRelocation info for .data section  nAddresses of pointer data that will need to be modified in the merged executable  .debug section  nInfo for symbolic debugging (gcc  int x = 1;  iint y = 2;  void a()  {  x = x+y;  printf("Total = %d\n",x);  }  int main(){a();}  080483c4 <a>:  80483c4: push %ebp  80483c5: mov %esp,%ebp  80483c7: sub $0x18,%esp  80483ca: movl $0x1,-0x8(%ebp)  80483d1: movl $0x2,-0x4(%ebp)  80483d8: mov -0x4(%ebp),%eax  80483db: add %eax,-0x8(%ebp)  80483de: mov -0x8(%ebp),%eax  80483e1: mov %eax,0x4(%esp)  80483e5: movl $0x80484f0,(%esp)  80483ec: call 80482dc <printf@plt>  80483f1: leave  80483f2: ret |
| Local Variables  void a()  {  int x = 1;  int y = 2;  x = x+y;  printf("Total = %d\n",x);  }  int main() {a();}  080483c4 <a>:  80483c4: push %ebp  80483c5: mov %esp,%ebp  80483c7: sub $0x8,%esp  80483ca: mov 0x804966c,%edx  80483d0: mov 0x8049670,%eax  80483d5: lea (%edx,%eax,1),%eax  80483d8: mov %eax,0x804966c  80483dd: mov 0x804966c,%eax  80483e2: mov %eax,0x4(%esp)  80483e6: movl $0x80484f0,(%esp)  80483ed: call 80482dc <printf@plt>  80483f2: leave  80483f3: ret |

|  |  |
| --- | --- |
| **If Statements**  void f(){  int x = 1;  int y = 2;  if (x==y) {  printf("x equals y.\n");  } else {  printf("x is not equal to y.\n");  }  }  int main() { f();} | 080483c4 <f>:  80483c4: push %ebp  80483c5: mov %esp,%ebp  80483c7: sub $0x18,%esp  80483ca: movl $0x1,-0x8(%ebp)  80483d1: movl $0x2,-0x4(%ebp)  80483d8: mov -0x8(%ebp),%eax  80483db: cmp -0x4(%ebp),%eax  80483de: jne 80483ee <f+0x2a>  80483e0: movl $0x80484f0,(%esp)  80483e7: call 80482d8 <puts@plt>  80483ec: jmp 80483fa <f+0x36>  80483ee: movl $0x80484fc,(%esp)  80483f5: call 80482d8 <puts@plt>  80483fa: leave  80483fb: ret |