REVERSE ENGINEERING

CHALLENGES

PWN COLLEGE ASSEMBLY MODULE

BY

JAIFIN B ALOOR

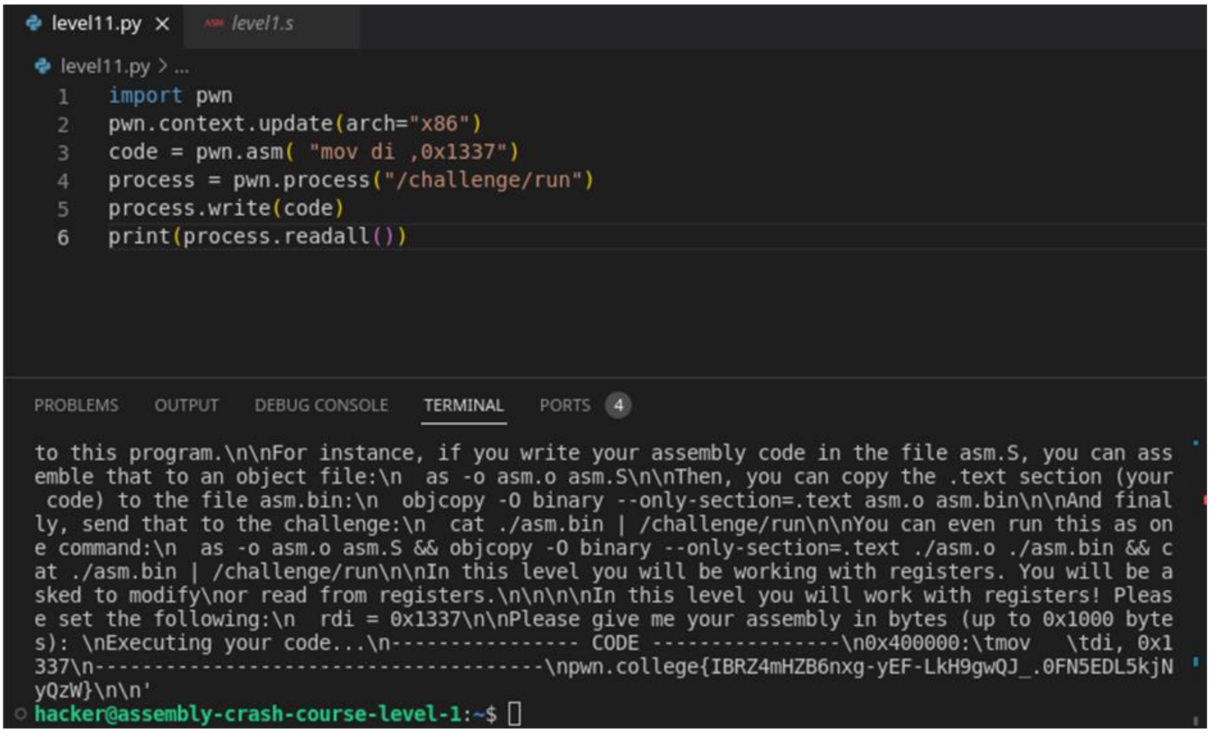
Level 1

The challenge of level1 is to set a register. First i wrote the code in python. I included the pwn module in python.

Then i set the architecture to x86 as the active context for current binary. Then inside pwn.asm(), i wrote the commands to solve the challenge and get the flag. Then i made a pwn process located at /challenges/run which will be used to interact with the binary. Then i actually run the code in binary and print the output of the process.

In this challenge i used the command mov di, 0x1337 to get the flag. The command moves the integer 1337 to the

Destination register.

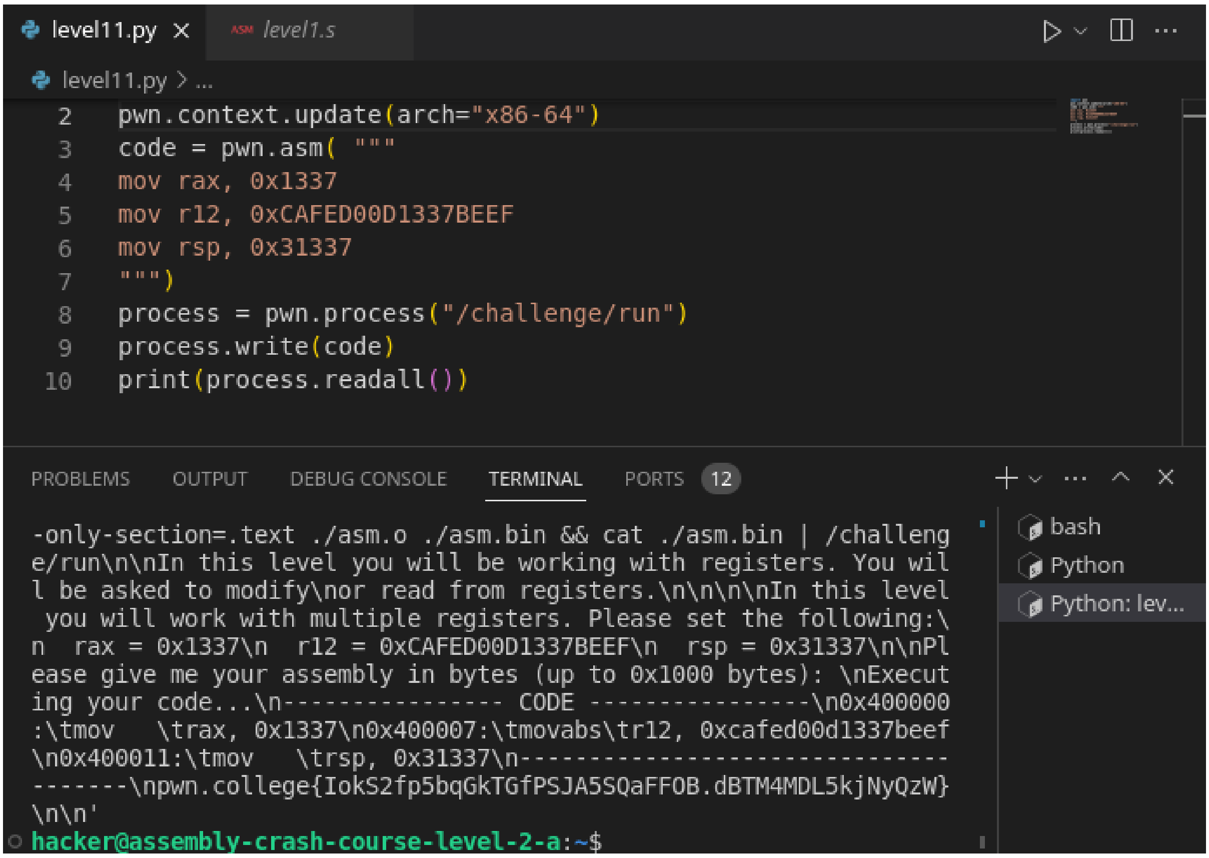


```python
import pwn
pwn.context.update(arch="x86")
code = pwn.asm( "mov di ,0x1337")
process = pwn.process("/challenge/run")
process.write(code)
print(process.readall())
```

```
to this program.\n\nFor instance, if you write your assembly code in the file asm.S, you can ass
emble that to an object file:\n  as -o asm.o asm.S\n\nThen, you can copy the .text section (your
 code) to the file asm.bin:\n  objcopy -O binary --only-section=.text asm.o asm.bin\n\nAnd final
ly, send that to the challenge:\n  cat ./asm.bin | /challenge/run\n\nYou can even run this as on
e command:\n  as -o asm.o asm.S && objcopy -O binary --only-section=.text ./asm.o ./asm.bin && c
at ./asm.bin | /challenge/run\n\nIn this level you will be working with registers. You will be a
sked to modify\nor read from registers.\n\n\n\nIn this level you will work with registers! Pleas
e set the following:\n  rdi = 0x1337\n\nPlease give me your assembly in bytes (up to 0x1000 byte
s): \nExecuting your code...\n---------------- CODE ----------------\n0x400000:\tmov   \tdi, 0x1
337\n--------------------------------------\npwn.college{IBRZ4mHZB6nxg-yEF-LkH9gwQJ_.0FN5EDL5kjN
yQzW}\n\n'
hacker@assembly-crash-course-level-1:~$ 
```

Level 2

I changed the architecture from x86 to x86-64. And i wrote the commands to set the registers one by one. Its very similar to the first challenge.

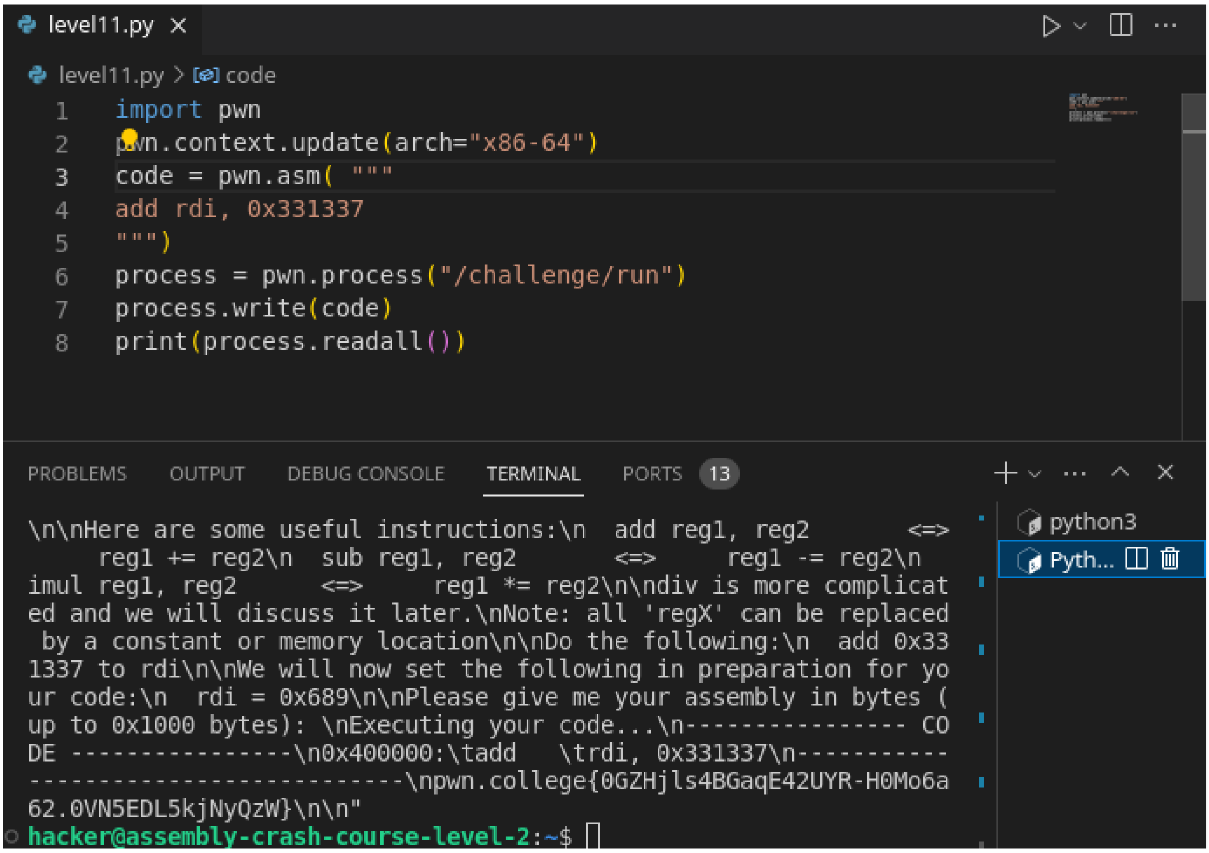


```python
pwn.context.update(arch="x86-64")
code = pwn.asm( """
mov rax, 0x1337
mov r12, 0xCAFED00D1337BEEF
mov rsp, 0x31337
""")
process = pwn.process("/challenge/run")
process.write(code)
print(process.readall())
```

```
-only-section=.text ./asm.o ./asm.bin && cat ./asm.bin | /challeng
e/run\n\nIn this level you will be working with registers. You wil
l be asked to modify\nor read from registers.\n\n\n\nIn this level
 you will work with multiple registers. Please set the following:\
n  rax = 0x1337\n  r12 = 0xCAFED00D1337BEEF\n  rsp = 0x31337\n\nPl
ease give me your assembly in bytes (up to 0x1000 bytes): \nExecut
ing your code...\n---------------- CODE ----------------\n0x400000
:\tmov   \trax, 0x1337\n0x400007:\tmovabs\tr12, 0xcafed00d1337beef
\n0x400011:\tmov   \trsp, 0x31337\n--------------------------------
-------\npwn.college{IokS2fp5bqGkTGfPSJA5SQaFFOB.dBTM4MDL5kjNyQzW}
\n\n'
hacker@assembly-crash-course-level-2-a:~$
```

Level 3

i used the command add rdi, 0x331337. which adds 331337 to the rdi and i got the flag.

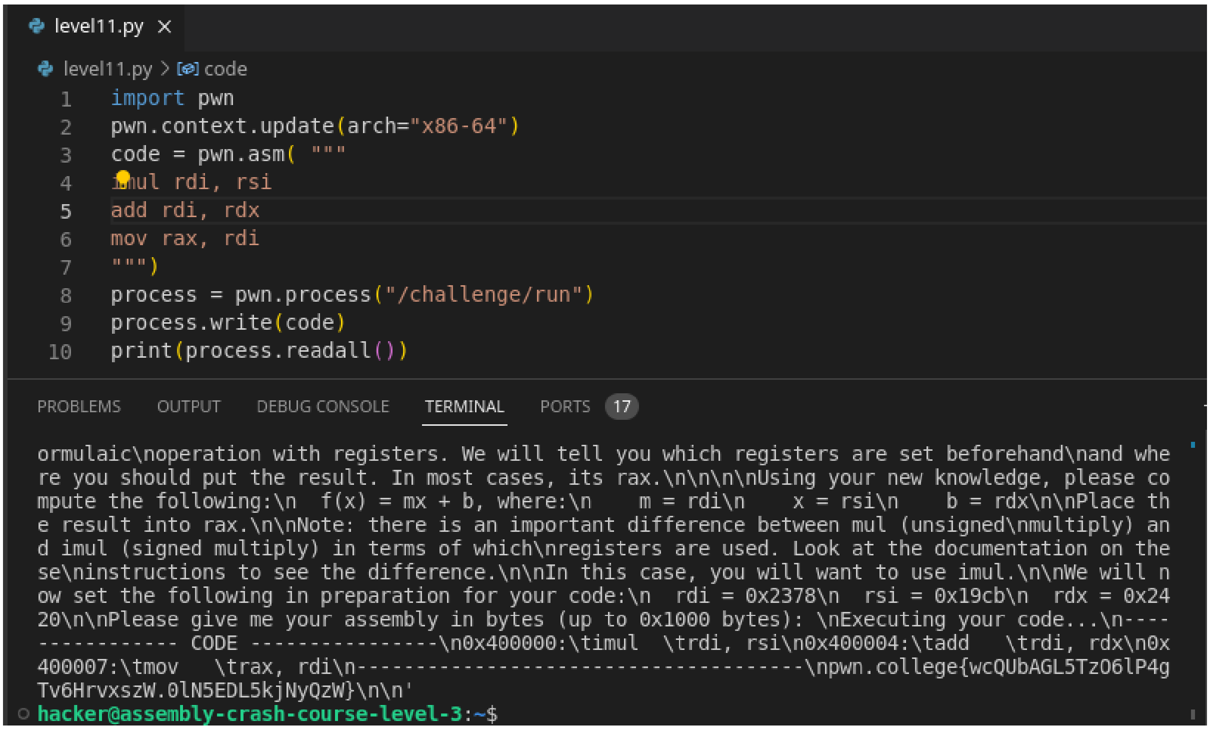


```python
import pwn
pwn.context.update(arch="x86-64")
code = pwn.asm( """
add rdi, 0x331337
""")
process = pwn.process("/challenge/run")
process.write(code)
print(process.readall())
```

```
\n\nHere are some useful instructions:\n  add reg1, reg2       <=>
     reg1 += reg2\n  sub reg1, reg2       <=>     reg1 -= reg2\n
imul reg1, reg2      <=>     reg1 *= reg2\n\ndiv is more complicat
ed and we will discuss it later.\nNote: all 'regX' can be replaced
 by a constant or memory location\n\nDo the following:\n  add 0x33
1337 to rdi\n\nWe will now set the following in preparation for yo
ur code:\n  rdi = 0x689\n\nPlease give me your assembly in bytes (
up to 0x1000 bytes): \nExecuting your code...\n---------------- CO
DE ----------------\n0x400000:\tadd   \trdi, 0x331337\n-----------
--------------------------\npwn.college{0GZHjls4BGaqE42UYR-H0Mo6a
62.0VN5EDL5kjNyQzW}\n\n"
hacker@assembly-crash-course-level-2:~$
```

Level 4

I used the imul command to multiply rdi(m) and rsi(x) first. Then i added the rdi(now mx) and rax(b). Finally i moved the rdi(now mx+b) to rax and i got the flag.



```python
import pwn
pwn.context.update(arch="x86-64")
code = pwn.asm( """
imul rdi, rsi
add rdi, rdx
mov rax, rdi
""")
process = pwn.process("/challenge/run")
process.write(code)
print(process.readall())
```

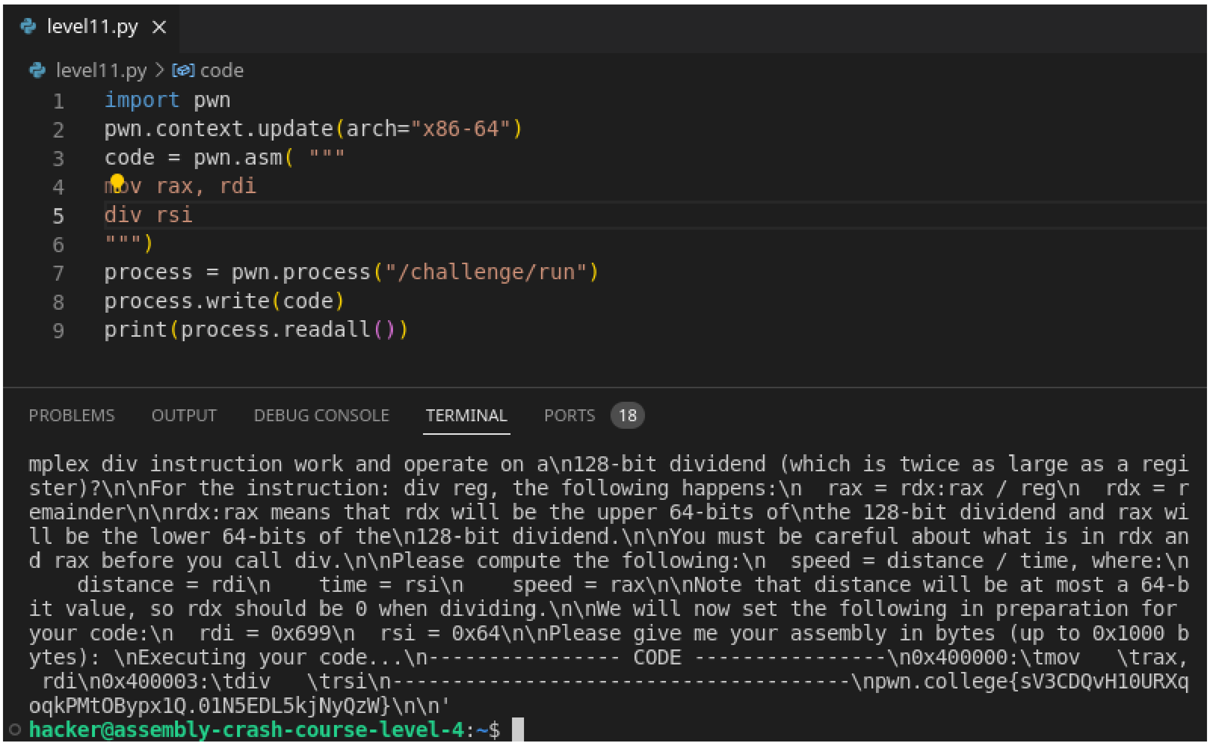
```
ormulaic\noperation with registers. We will tell you which registers are set beforehand\nand whe
re you should put the result. In most cases, its rax.\n\n\n\nUsing your new knowledge, please co
mpute the following:\n  f(x) = mx + b, where:\n    m = rdi\n    x = rsi\n    b = rdx\n\nPlace th
e result into rax.\n\nNote: there is an important difference between mul (unsigned\nmultiply) an
d imul (signed multiply) in terms of which\nregisters are used. Look at the documentation on the
se\ninstructions to see the difference.\n\nIn this case, you will want to use imul.\n\nWe will n
ow set the following in preparation for your code:\n  rdi = 0x2378\n  rsi = 0x19cb\n  rdx = 0x24
20\n\nPlease give me your assembly in bytes (up to 0x1000 bytes): \nExecuting your code...\n----
------------ CODE ----------------\n0x400000:\timul  \trdi, rsi\n0x400004:\tadd   \trdi, rdx\n0x
400007:\tmov   \trax, rdi\n--------------------------------------\npwn.college{wcQUbAGL5TzO6lP4g
Tv6HrvxszW.0lN5EDL5kjNyQzW}\n\n'
hacker@assembly-crash-course-level-3:~$
```

Level 5

First i moved rdi to rax with the command mov rax, rdi . Then i divided the value by rsi(time) to get speed.

Rdx:rax means that the upper 64 bits of the 128 bit divident will be stored in the rdx and the lower 64 bits will be

Stored in the rax.

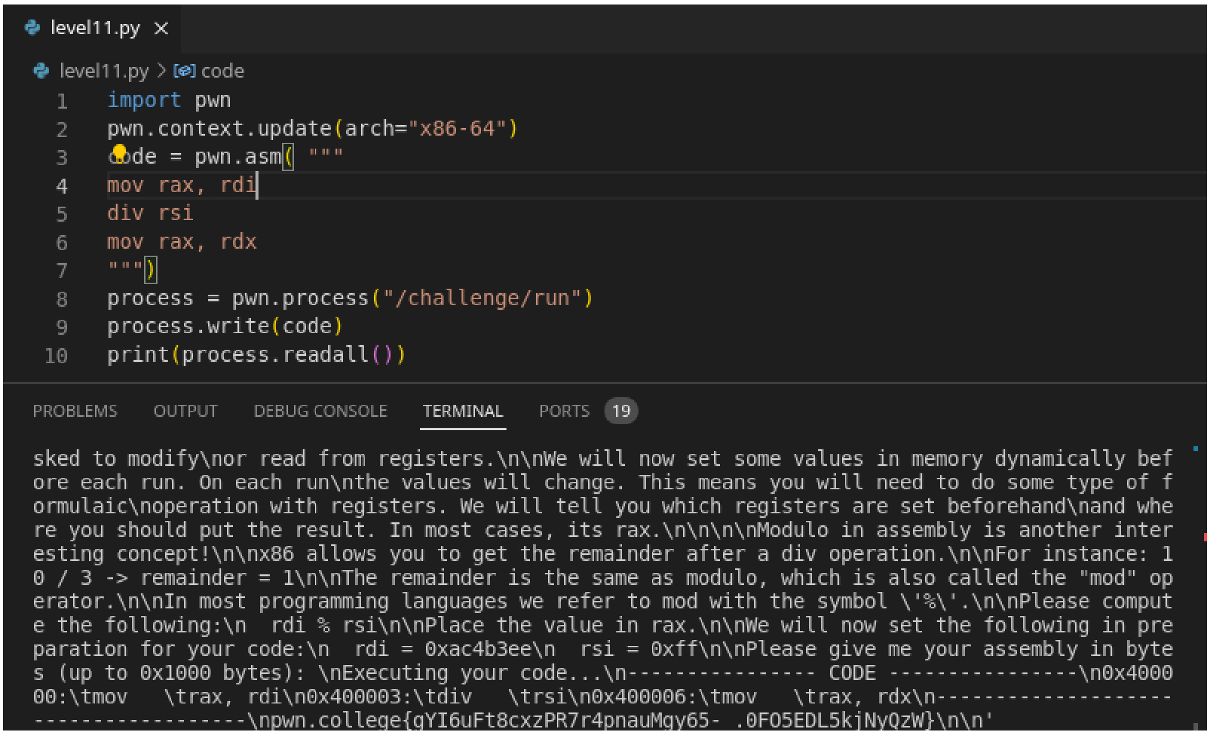


```python
import pwn
pwn.context.update(arch="x86-64")
code = pwn.asm( """
mov rax, rdi
div rsi
""")
process = pwn.process("/challenge/run")
process.write(code)
print(process.readall())
```

```
mplex div instruction work and operate on a\n128-bit dividend (which is twice as large as a regi
ster)?\n\nFor the instruction: div reg, the following happens:\n  rax = rdx:rax / reg\n  rdx = r
emainder\n\nrdx:rax means that rdx will be the upper 64-bits of\nthe 128-bit dividend and rax wi
ll be the lower 64-bits of the\n128-bit dividend.\n\nYou must be careful about what is in rdx an
d rax before you call div.\n\nPlease compute the following:\n  speed = distance / time, where:\n
    distance = rdi\n    time = rsi\n    speed = rax\n\nNote that distance will be at most a 64-b
it value, so rdx should be 0 when dividing.\n\nWe will now set the following in preparation for
your code:\n  rdi = 0x699\n  rsi = 0x64\n\nPlease give me your assembly in bytes (up to 0x1000 b
ytes): \nExecuting your code...\n---------------- CODE ----------------\n0x400000:\tmov   \trax,
 rdi\n0x400003:\tdiv   \trsi\n--------------------------------------\npwn.college{sV3CDQvH10URXq
oqkPMtOBypx1Q.01N5EDL5kjNyQzW}\n\n'
hacker@assembly-crash-course-level-4:~$
```

Level 6

First i moved the rd i(divident) to rax using mov command. Then i divided it by rsi using div. To get the remainder in rax i finally moved the rdx into the rax register.

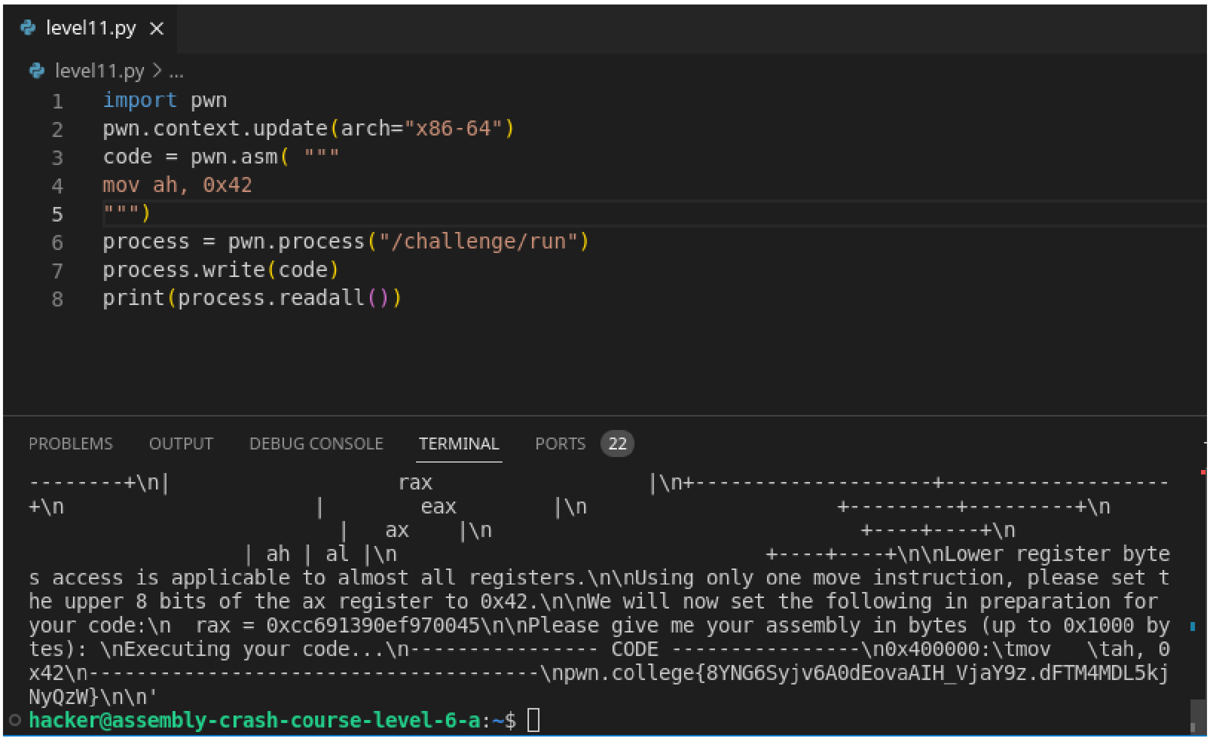


```python
import pwn
pwn.context.update(arch="x86-64")
code = pwn.asm( """
mov rax, rdi
div rsi
mov rax, rdx
""")
process = pwn.process("/challenge/run")
process.write(code)
print(process.readall())
```

```
sked to modify\nor read from registers.\n\nWe will now set some values in memory dynamically bef
ore each run. On each run\nthe values will change. This means you will need to do some type of f
ormulaic\noperation with registers. We will tell you which registers are set beforehand\nand whe
re you should put the result. In most cases, its rax.\n\n\n\nModulo in assembly is another inter
esting concept!\n\nx86 allows you to get the remainder after a div operation.\n\nFor instance: 1
0 / 3 -> remainder = 1\n\nThe remainder is the same as modulo, which is also called the "mod" op
erator.\n\nIn most programming languages we refer to mod with the symbol \'%\'.\n\nPlease comput
e the following:\n  rdi % rsi\n\nPlace the value in rax.\n\nWe will now set the following in pre
paration for your code:\n  rdi = 0xac4b3ee\n  rsi = 0xff\n\nPlease give me your assembly in byte
s (up to 0x1000 bytes): \nExecuting your code...\n---------------- CODE ----------------\n0x4000
00:\tmov   \trax, rdi\n0x400003:\tdiv   \trsi\n0x400006:\tmov   \trax, rdx\n--------------------
------------------\npwn.college{gYI6uFt8cxzPR7r4pnauMgy65- .0FO5EDL5kjNyQzW}\n\n'
```

Level 7

i used the mov command to move the 42 to the ah(the upper 8 bits of the ax register). and i got the flag.



```python
import pwn
pwn.context.update(arch="x86-64")
code = pwn.asm( """
mov ah, 0x42
""")
process = pwn.process("/challenge/run")
process.write(code)
print(process.readall())
```

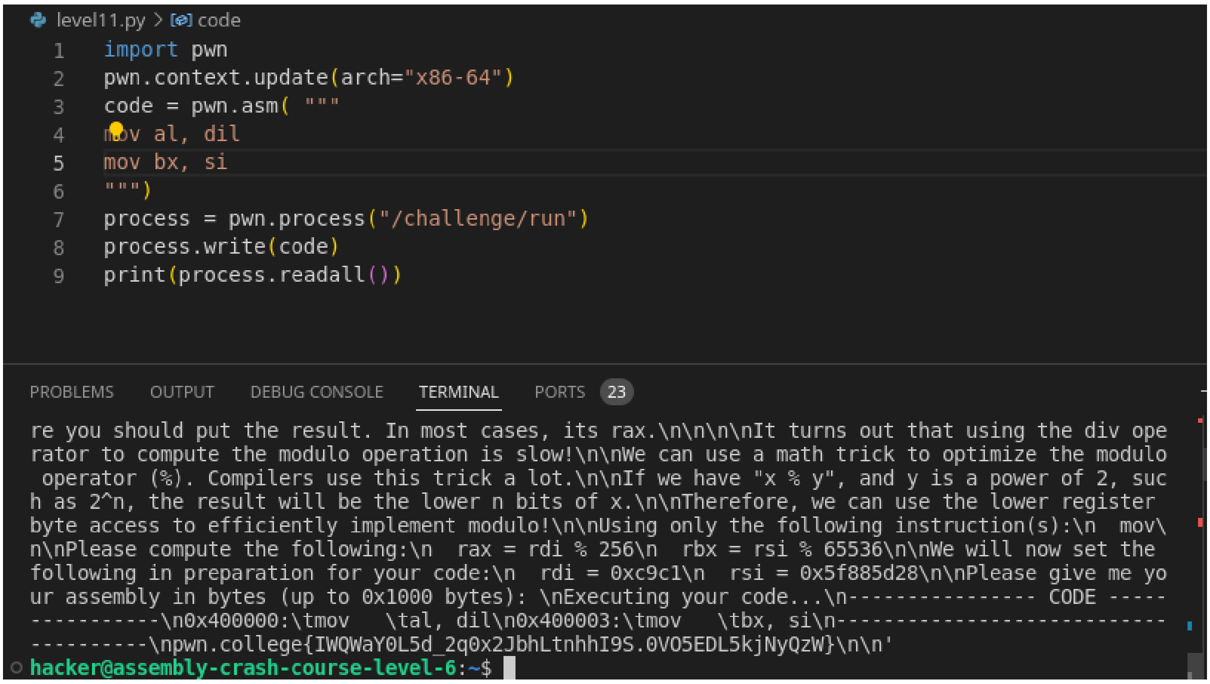
```
--------+\n|                    rax                    |\n+--------------------+------------------
+\n                     |         eax         |\n                     +---------+---------+\n
                          |   ax    |\n                              +----+----+\n
                  | ah | al |\n                              +----+----+\n\nLower register byte
s access is applicable to almost all registers.\n\nUsing only one move instruction, please set t
he upper 8 bits of the ax register to 0x42.\n\nWe will now set the following in preparation for
your code:\n  rax = 0xcc691390ef970045\n\nPlease give me your assembly in bytes (up to 0x1000 by
tes): \nExecuting your code...\n---------------- CODE ----------------\n0x400000:\tmov   \tah, 0
x42\n--------------------------------------\npwn.college{8YNG6Syjv6A0dEovaAIH_VjaY9z.dFTM4MDL5kj
NyQzW}\n\n'
○ hacker@assembly-crash-course-level-6-a:~$ 
```

Level 8

If we have "x % y", and y is a power of 2, such as 2^n, the result will be the lower n bits of x.

Therefore, we can use the lower register byte access to efficiently implement modulo. I used mov command to move the

Dil(the lower 8 bits of the rdi register) to the al(the lower 8 bits of rax). Again i used the mov command to move the si(the lower 16 bits of the rsi register) to bx(the lower 16 bits of the rbx register). And i got the flag.

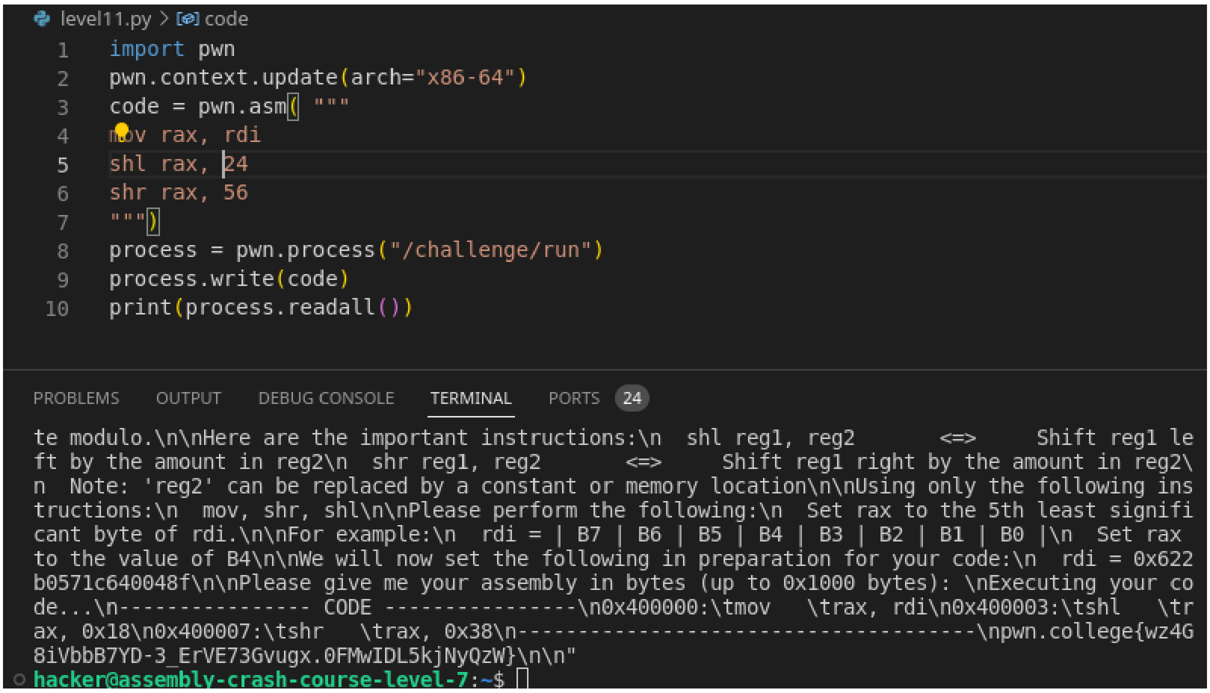


```python
import pwn
pwn.context.update(arch="x86-64")
code = pwn.asm( """
mov al, dil
mov bx, si
""")
process = pwn.process("/challenge/run")
process.write(code)
print(process.readall())
```

```
re you should put the result. In most cases, its rax.\n\n\n\nIt turns out that using the div ope
rator to compute the modulo operation is slow!\n\nWe can use a math trick to optimize the modulo
 operator (%). Compilers use this trick a lot.\n\nIf we have "x % y", and y is a power of 2, suc
h as 2^n, the result will be the lower n bits of x.\n\nTherefore, we can use the lower register
byte access to efficiently implement modulo!\n\nUsing only the following instruction(s):\n  mov\
n\nPlease compute the following:\n  rax = rdi % 256\n  rbx = rsi % 65536\n\nWe will now set the
following in preparation for your code:\n  rdi = 0xc9c1\n  rsi = 0x5f885d28\n\nPlease give me yo
ur assembly in bytes (up to 0x1000 bytes): \nExecuting your code...\n---------------- CODE -----
-----------\n0x400000:\tmov   \tal, dil\n0x400003:\tmov   \tbx, si\n----------------------------
----------\npwn.college{IWQWaY0L5d_2q0x2JbhLtnhhI9S.0VO5EDL5kjNyQzW}\n\n'
hacker@assembly-crash-course-level-6:~$
```

Level 9

Use shr and shl to shift the values to the right and left. By the number of bits in the second argument.first i moved the value in rdi to rax. Then i shifted the binary to the left by 24 bits in order to get rid to the bytes more signifiacnt than the 5th most lsv. Then i shifted the current binary to the right by 56 bits to get rid of the lesser significant digits. I used the mov , shl and shr commands.

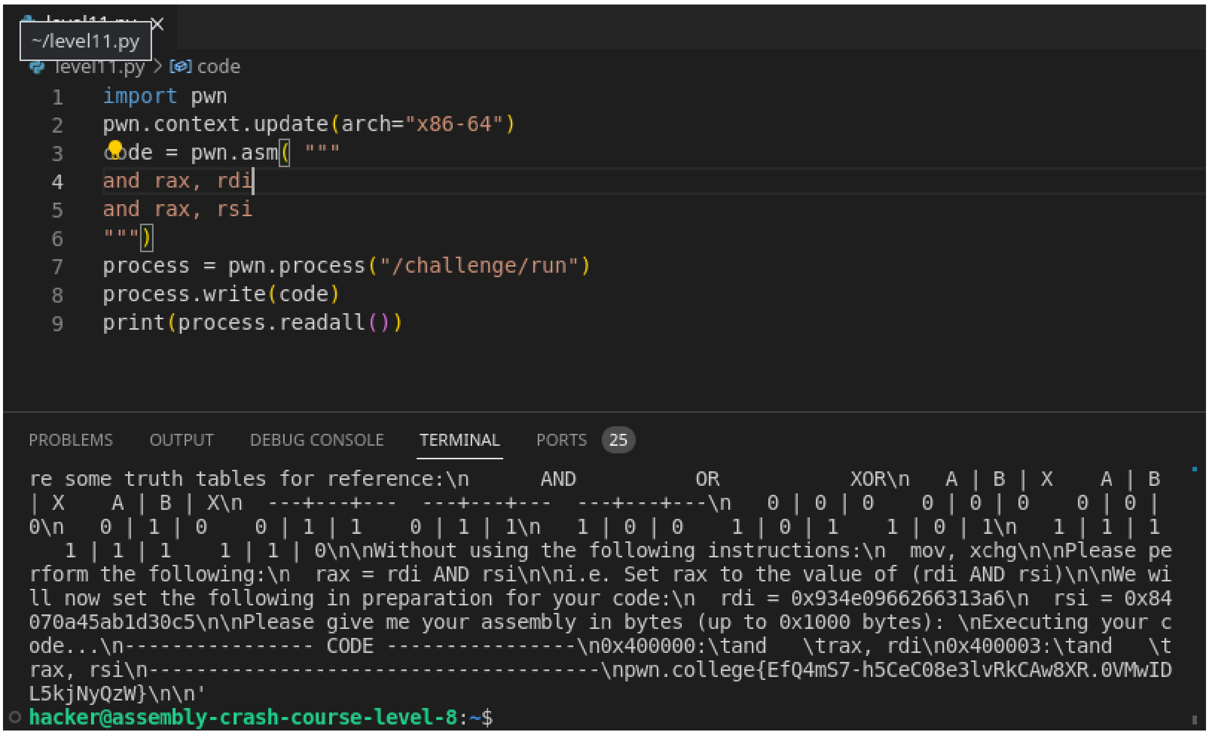


```python
import pwn
pwn.context.update(arch="x86-64")
code = pwn.asm( """
mov rax, rdi
shl rax, 24
shr rax, 56
""")
process = pwn.process("/challenge/run")
process.write(code)
print(process.readall())
```

```
te modulo.\n\nHere are the important instructions:\n  shl reg1, reg2       <=>     Shift reg1 le
ft by the amount in reg2\n  shr reg1, reg2       <=>     Shift reg1 right by the amount in reg2\
n  Note: 'reg2' can be replaced by a constant or memory location\n\nUsing only the following ins
tructions:\n  mov, shr, shl\n\nPlease perform the following:\n  Set rax to the 5th least signifi
cant byte of rdi.\n\nFor example:\n  rdi = | B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 |\n  Set rax
to the value of B4\n\nWe will now set the following in preparation for your code:\n  rdi = 0x622
b0571c640048f\n\nPlease give me your assembly in bytes (up to 0x1000 bytes): \nExecuting your co
de...\n---------------- CODE ----------------\n0x400000:\tmov   \trax, rdi\n0x400003:\tshl   \tr
ax, 0x18\n0x400007:\tshr   \trax, 0x38\n--------------------------------------\npwn.college{wz4G
8iVbbB7YD-3_ErVE73Gvugx.0FMwIDL5kjNyQzW}\n\n"
hacker@assembly-crash-course-level-7:~$ 
```

Level 10

And, or, not & xor are the key logical operators in x86. If we apply a logical operator in a binary the result will be calculated bit by bit, that's why its called bitwise logic. First i used the and operator with rax and rdi, since rax = rdi and rsi, rax and rdi = rdi. So rdi is moved to rax without using mov. Then i anded the new rax and rsi to get the flag.



```
import pwn
pwn.context.update(arch="x86-64")
code = pwn.asm( """
and rax, rdi
and rax, rsi
""")
process = pwn.process("/challenge/run")
process.write(code)
print(process.readall())
```

```
re some truth tables for reference:\n      AND           OR            XOR\n   A | B | X    A | B
| X    A | B | X\n  ---+---+---  ---+---+---  ---+---+---\n   0 | 0 | 0    0 | 0 | 0    0 | 0 |
0\n   0 | 1 | 0    0 | 1 | 1    0 | 1 | 1\n   1 | 0 | 0    1 | 0 | 1    1 | 0 | 1\n   1 | 1 | 1
   1 | 1 | 1    1 | 1 | 0\n\nWithout using the following instructions:\n  mov, xchg\n\nPlease pe
rform the following:\n  rax = rdi AND rsi\n\ni.e. Set rax to the value of (rdi AND rsi)\n\nWe wi
ll now set the following in preparation for your code:\n  rdi = 0x934e0966266313a6\n  rsi = 0x84
070a45ab1d30c5\n\nPlease give me your assembly in bytes (up to 0x1000 bytes): \nExecuting your c
ode...\n---------------- CODE ----------------\n0x400000:\tand   \trax, rdi\n0x400003:\tand   \t
rax, rsi\n--------------------------------------\npwn.college{EfQ4mS7-h5CeC08e3lvRkCAw8XR.0VMwID
L5kjNyQzW}\n\n'
hacker@assembly-crash-course-level-8:~$
```

PWN COLLEGE ASSEMBLY CHALLENGES FLAGS

level1 = pwn.college{IBRZ4mHZB6nxg-yEF-LkH9gwQJ_.0FN5EDL5kjNyQzW}

level2 = pwn.college{IokS2fp5bqGkTGfPSJA5SQaFFOB.dBTM4MDL5kjNyQzW}

level3 = pwn.college{0GZHjls4BGaqE42UYR-H0Mo6a62.0VN5EDL5kjNyQzW}

level4 = pwn.college{wcQUbAGL5TzO6lP4gTv6HrvxszW.0lN5EDL5kjNyQzW}

level5 = pwn.college{sV3CDQvH10URXqoqkPMtOBypx1Q.01N5EDL5kjNyQzW}

level6 = pwn.college{gYI6uFt8cxzPR7r4pnauMgy65-_.0FO5EDL5kjNyQzW}

level7 = pwn.college{8YNG6Syjv6A0dEovaAIH_VjaY9z.dFTM4MDL5kjNyQzW}

level8 = pwn.college{IWQWaY0L5d_2q0x2JbhLtnhhI9S.0VO5EDL5kjNyQzW}

level9 = pwn.college{wz4G8iVbbB7YD-3_ErVE73Gvugx.0FMwIDL5kjNyQzW}

level10= pwn.college{EfQ4mS7-h5CeC08e3lvRkCAw8XR.0VMwIDL5kjNyQzW}