Embedded Programmer 과정
Homework days 3

김형주

Ex1.

스키장에서 스키 장비를 임대하는데 37500원이 든다.
또 3일 이상 이용할 경우 20%를 할인 해준다.
일주일간 이용할 경우 임대 요금은 얼마일까 ?
(연산 과정은 모두 함수로 돌린다)

```c
/*
        description :
        program to calculate rental fare

        created by HyungjuKim
        contact : 010-9132-6404(kr), mihaelkel@naver.com
*/

#include <stdio.h>
int calculateFare(int days);

int main(void){
        int days = 7, cost;

        cost = calculateFare(days);
        printf("cost : %d\n",cost);

        return 0;
}

int calculateFare(int days){

        int day = 1;
        int cost_a_day = 37500;
        float rate = 0.2;
        int rateDay = 3;
        int totalCost = 0;
        while(day <= days){
                if(day == rateDay)
                        cost_a_day*=(1-rate);
                totalCost+=cost_a_day;
                day++;
        }
        return totalCost;
}
```

```
howard@ubuntu:~/Mytest/Mytest$ ./ex1
cost : 225000
```

Ex2.

어떤 수의 약수를 구하는 함수를 작성하여 출력하시오.

howard@ubuntu: ~/Mytest/Mytest

```c
/*
        description :
        program to find all divisors of input number

        created by HyungjuKim
        contact : 010-9132-6404(kr), mihaelkel@naver.com
*/

#include <stdio.h>
void FindDivisorFunc(int num);

int main(void){
        int num;

        printf("input number : ");
        scanf("%d",&num);

        FindDivisorFunc(num);

        return 0;
}

void FindDivisorFunc(int num){
        int i = 1, mid = (int)num/2;
        printf("divisors : ");
        while(i<=mid){
                if(!(num%i))
                        printf("%d ",i);
                i++;
        }
        printf("\n");

}
```

```
howard@ubuntu:~/Mytest/Mytest$ ./ex2
input number : 100
divisors : 1 2 4 5 10 20 25 50
```

Ex3.

1 ~ 1000사이에 3의 배수의 합을 구하시오.

howard@ubuntu: ~/Mytest/Mytest

```c
/*
        description :
        program to calculate sum of multiples of 3 range from 1 to 1000

        created by HyungjuKim
        contact : 010-9132-6404(kr), mihaelkel@naver.com
*/

#include <stdio.h>
void SumFunc(int start,int end,int multiple){
        int i = start, sum = 0;
        while(i<=end){
                if(!(i%multiple)){
                        sum+=i;
                }
                i++;
        }
        printf("sum multiples of %d (%d~%d) : %d\n",multiple, start, end, sum);

}
int main(void){
        int start = 1, end = 1000, multiple = 3;
        SumFunc(start,end,multiple);


        return 0;
}
```

```
howard@ubuntu:~/Mytest/Mytest$ ./ex2
sum multiples of 3 (1~10) : 18
```

Ex4.

1 ~ 1000사이에 4나 6으로 나눠도 나머지가 1인 수의 합을 출력하라

howard@ubuntu: ~/Mytest/Mytest

```
/*
        description :
        program to calculate sum of numbers that the rest be 1 when divided by 4 or 6,
        range from 1 to 1000

        created by HyungjuKim
        contact : 010-9132-6404(kr), mihaelkel@naver.com
*/

#include <stdio.h>
void SumFunc(int start,int end,int quotient1,int quotient2,int remainder){
        int i = start, sum = 0;
        while(i<=end){
                if(!(i%quotient1-1)||!(i%quotient2-1)){ // i%4==1 or i%6==1
                        sum+=i;
                }
                i++;
        }
        printf("sum  of conditioned number : %d\n",sum);

}
int main(void){
        int start = 1, end = 1000, quotient1 = 4, quotient2 = 6, remainder = 1;
        SumFunc(start,end,quotient1,quotient2,remainder);


        return 0;
}
```

```
howard@ubuntu:~/Mytest/Mytest$ ./ex4
sum  of conditioned number : 166167
```

Ex5.

7의 배수로 이루어진 값들이 나열되어 있다고 가정한다.
함수의 인자(input)로 항의 갯수를 받아서 마지막 항의 값을 구하는 프로그램을 작성하라.

```c
/*
        description :
        program to find Nth(input) term of the sequence that a(n) = 7*n.

        created by HyungjuKim
        contact : 010-9132-6404(kr), mihaelkel@naver.com
*/

#include <stdio.h>
void FindTermFunc(int n,int ratio);
int main(void){
        int n, ratio = 7;
        printf("Input n :");
        scanf("%d",&n);
        FindTermFunc(n, ratio);

        return 0;
}
void FindTermFunc(int n,int ratio){
        int i = 1, an = 0;

        while(i<=n){
                an+=ratio;
                i++;
        }
        printf("%dth term of a(n) : %d\n",n,an);
}
```

```
howard@ubuntu:~/Mytest/Mytest$ ./ex5
Input n :100
100th term of a(n) : 700
```

Ex6.

Little Endian과 Big Endian의 차이를 기술하시오.
(최소 100자 이상 기술하시오 - 성능적인 측면 관점등)

0x00 00 00 03 표기

빅 엔디안과 리틀 엔디안은 Byte Order의 정렬 방법을 의미한다. Big Endian 방식은 큰 단위가 앞에 오는 방식이고, Little Endian 방식은 작은 단위가 앞에 오는 방식이다. 예를 들면, 0x50 F1 FA 03 이라는 데이터가 있을 때, Big Endian 방식은 우리가 읽는 그대로, 0x50 F1 FA 03이 되고, Little Endian 방식은 0x03 FA F1 50이 된다.

빅 엔디안	
00	0x2000
00	0x2008
00	0x2010
03	0x2018

리틀 엔디안	
03	0x2000
00	0x2008
00	0x2010
00	0x2018

Big Endian 방식을 사용하면, 사람이 보기 편하다. 즉, 디버깅하기가 쉽다. Little Endian 방식을 사용하면, 디버깅하기는 다소 불편할 수도 있지만, 속도 측면에서 이득을 볼 수 있다.
0x00 00 00 03이라는 데이터가 있다고 가정하다. 이 데이터가 홀수인지 짝수인지 판별한다고 할 때, Big Endian 방식으로 판별하기 위해서는 주소를 4번 불러와야 한다(0x2000, 0x2008, 0x2010, 0x2018). 4번째 주소를 불러온 후, 그 주소에 있는 데이터를 읽은 다음 홀수or짝수 판별을 해야한다. Little Endian 방식의 경우, 1번째 주소(0x2000)만 불러온 후 그 주소에 있는 데이터를 읽고 연산하면 된다.

Ex7.

C로 함수를 만들 때, Stack이란 구조가 생성된다. 이 구조가 어떻게 동작하는지 Assembly Language를 해석하며 기술해보시오. esp, ebp, eip등의 Register에 어떤 값이 어떻게 들어가는지 등 메모리에 어떤 값들이 들어가는지 등을 자세히 기술하시오.

```c
/*
        description :
        program to analize architecture of Stack when a fuction made

        created by HyungjuKim
        contact : 010-9132-6404(kr), mihaelkel@naver.com
*/

#include <stdio.h>
int multi2(int num);
int main(){
        int i, sum = 0, result;
        for(i = 0;i < 5;i++){
                sum+=i;
        }
        result = multi2(sum);
        printf("result = %d\n",result);
        return 0;
}
int multi2(int num){
        return num*2;
}
```

디버깅 과정은 맨 뒤.

```
Dump of assembler code for function main:
=> 0x0000000000400526 <+0>:     push   %rbp
   0x0000000000400527 <+1>:     mov    %rsp,%rbp
   0x000000000040052a <+4>:     sub    $0x10,%rsp
   0x000000000040052e <+8>:     movl   $0x0,-0x8(%rbp)
   0x0000000000400535 <+15>:    movl   $0x0,-0xc(%rbp)
   0x000000000040053c <+22>:    jmp    0x400548 <main+34>
   0x000000000040053e <+24>:    mov    -0xc(%rbp),%eax
   0x0000000000400541 <+27>:    add    %eax,-0x8(%rbp)
   0x0000000000400544 <+30>:    addl   $0x1,-0xc(%rbp)
   0x0000000000400548 <+34>:    cmpl   $0x4,-0xc(%rbp)
   0x000000000040054c <+38>:    jle    0x40053e <main+24>
   0x000000000040054e <+40>:    mov    -0x8(%rbp),%eax
   0x0000000000400551 <+43>:    mov    %eax,%edi
   0x0000000000400553 <+45>:    callq  0x400576 <multi2>
   0x0000000000400558 <+50>:    mov    %eax,-0x4(%rbp)
   0x000000000040055b <+53>:    mov    -0x4(%rbp),%eax
   0x000000000040055e <+56>:    mov    %eax,%esi
   0x0000000000400560 <+58>:    mov    $0x400614,%edi
   0x0000000000400565 <+63>:    mov    $0x0,%eax
   0x000000000040056a <+68>:    callq  0x400400 <printf@plt>
   0x000000000040056f <+73>:    mov    $0x0,%eax
   0x0000000000400574 <+78>:    leaveq
   0x0000000000400575 <+79>:    retq
End of assembler dump.
```

Ex8.

goto를 어떤 경우에 사용하는 것이 유용한지 기술하시오.

아래와 같은 이중 루프가 있다고 하자. 아래 프로그램은 이차원배열 arr[10][10]에서 그 값이 5인 요소를 검색하는 프로그램이다.

```
for(int i=0;i<10;i++){
    for(int j=0;j<10;j++){
        if(!(arr[i][j]-5)){
            res = arr[i][j];
            chk = 1;
            break;
        }
    }
    if(chk)
        break;
}
```

위와 같이, 이중 루프에서 빠져나오기 위해서는 break문을 2번 써야하고, 2번째 루프를 빠져나오기 위해 추가로 chk라는 확인 변수도 필요하다.
반면, goto문을 사용하면 한번에 빠져나올 수 있다.

```
for(int i=0;i<10;i++){
    for(int j=0;j<10;j++){
        if(!(arr[i][j]-5)){
            res = arr[i][j];
            goto LABLE:
        }
    }
}
LABLE:
```

Ex9.

static Keyword의 용도에 대해서 기술해보시오

main()함수에서 다른 함수를 재호출할 때, 그 함수에 있는 변수는 다시 초기화된다. 아래의 코드를 살펴보자.

```c
#include <stdio.h>
void StaticTest();
int main(){
        StaticTest();
        StaticTest();
        StaticTest();
        return 0;
}
void StaticTest(){
        int num = 3;
        num++;
        printf("%d\n",num);

}
```

```
howard@ubuntu:~/Mytest/Mytest$ ./test
4
4
4
```

StaticTest() 안에 있는 변수 num은, main()이 호출할 때마다 3으로 초기화된다. num의 데이터를 보존하고 싶을 때, static을 사용한다.

```c
#include <stdio.h>
void StaticTest();
int main(){
        StaticTest();
        StaticTest();
        StaticTest();
        return 0;
}
void StaticTest(){
        static int num = 3;
        num++;
        printf("%d\n",num);

}
```

```
howard@ubuntu:~/Mytest/Mytest$ ./test
4
5
6
```

위와 같이 static 으로 변수를 선언하면, 함수를 재호출할 때 그 전에 변수의 데이터값을 보존할 수 있다.

Ex10.

구구단을 만들어보시오

howard@ubuntu: ~/Mytest/Mytest

```
/*
        description :
        program to display multiplication table(multiples : 2 to 9)

        created by HyungjuKim
        contact : 010-9132-6404(kr), mihaelkel@naver.com
*/

#include <stdio.h>
void MakeTable(int nStart, int nEdn, int start, int end);
int main(void){
        int nStart = 2, nEnd = 9, start = 1, end = 9;

        MakeTable(nStart, nEnd, start, end);

        return 0;
}
void MakeTable(int nStart, int nEnd, int start, int end){
        int i, j = nStart;

        while(j<=nEnd){
                i = start;
                printf("%d step\n",j);
                while(i<=end){
                        printf("%d*%d=%d\n",j,i,j*i);
                        i++;
                }
                j++;
                printf("\n");
        }
}
```

```
howard@ubuntu:~/Mytest/Mytest$ ./ex10
2 step
2*1=2
2*2=4
2*3=6
2*4=8
2*5=10
2*6=12
2*7=14
2*8=16
2*9=18

3 step
3*1=3
3*2=6
3*3=9
3*4=12
3*5=15
3*6=18
3*7=21
3*8=24
3*9=27

4 step
4*1=4
4*2=8
4*3=12
4*4=16
4*5=20
4*6=24
4*7=28
4*8=32
4*9=36

5 step
5*1=5
5*2=10
5*3=15
```

Ex11.

for문, while문, if문, switch문등의 제어문이 각각 어떤 상황에서 가장 유용할 수 있는지 고려해보시오.

먼저 for문과 while문을 비교해보자.

```
for(int i=0;i<10;i++){
}
------------------
i=0;
while(i<10){
  i++;
}
```

for문의 경우, 선언과 동시에 초기화, 조건식, 증감식을 모두 요구한다. 즉, 초기화, 조건식, 증감식이 모두 필요할 경우 while문보다는 for문이 보기 편하다.

무한루프를 만든 후, 여러 가지 조건에 따라 반복문을 빠져나오는 경우는 while문이 보기 편하다.

```
while(1){
  switch(input){
      case 1:
          break;
      case 2:
          break;
  }
}
for(;;){
 switch input:
      case 1:
          break;
      case 2:
          break;
}
```

if문과 switch문을 비교해보자.

if문의 괄호()안에는 조건이 들어가고, switch문의 case 1: 에는 특정한 값이 들어간다. 즉, 비교연산식을 사용할 경우, if문이 유용하고 값이 정해져있는 경우 switch ~ case 문이 유용하다.

또한, 조건이 많을 경우 if문을 여러번 쓰는 것보다는 switch ~ case문을 쓰는 것이 보기가 편하다.

예를 들어, 정수를 입력 받은 후, 음수,양수,0을 판별하는 프로그램을 생각해보자

```
if(num>0)
    printf("양수\n");
else if(num<0)
    printf("음수\n");
else
    printf("0\n");
```

이렇게 조건이 비교 연산식이 필요한 경우, switch문보다는 if문이 편하다.

다음으로 switch문이 유용한 경우를 살펴보자.

게임을 구현한다고 가정해보자. 스킬 key는 q,w,e,r이라 했을 때, 스킬 사용함수를 구현하려면 (메뉴창 보기, Score 보기 등 다른 기능은 배제하고 스킬 구현만 생각한다.)

```
if(input_key == 'q')
     q_Skill();
       :
switch(input_key){
     case 'q':
         q_Skill();
         breal;
         :
}
```

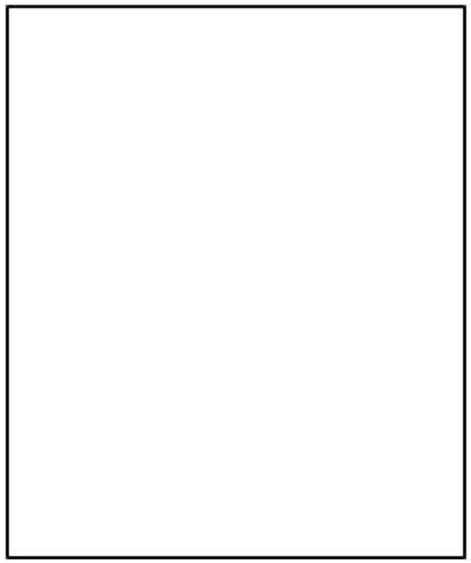
if문보다는 switch문 사용시 보기 편하다.

Ex12.

Linux에서 Debugging하는 방법에 대해 기술해보시오. Break Point는 어떻게 잡으며, 조사식, 메모리, 레지스터 등의 디버그 창은 각각 어떤 역할을 하고 무엇을 알고자 할 때 유용한지 기술하시오.

1. break point 잡는법 : b main , b* 0x7fffffdcd0 등 b명령어를 통해 잡는다.
2. disas 명령어를 통해 어셈블리어 코드를 볼 수 있고, p/x 명령어를 통해 해당 레지스터의 주소를, p 명령어를 통해 해당 레지스터에 저장된 데이터를 볼 수 있다.
3. si명령어를 통해 1단위(1레지스터)씩 프로그램을 실행시켜 프로그램이 의도한 데로 작동하는 지 확인할 수 있다.
4. rip(eip) 레지스터를 확인하면, 다음 실행할 레지스터를 알 수 있다.
5. rax(eax) 레지스터는 주로 산술 연산에 사용되므로, 결과값이 의도와 다르다면 이 레지스터를 주의깊게 보아 디버깅할 수 있다.

7번문제 디버깅 과정



0xdd18 rsp

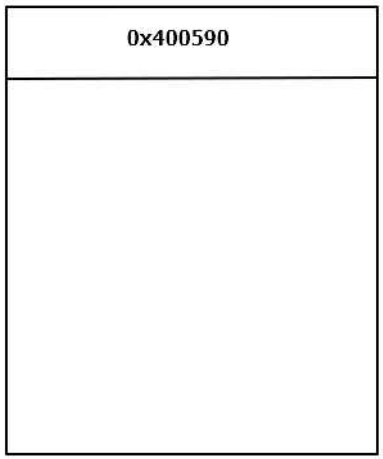
0x400590 rbp

Stack

rsp	0xdd18
rbp	0x400590
rip	0x400526 (push %rbp)
eax	0x400526
edi	0x1

Registers in CPU

```
Dump of assembler code for function main:
=> 0x0000000000400526 <+0>:     push   %rbp
   0x0000000000400527 <+1>:     mov    %rsp,%rbp
   0x000000000040052a <+4>:     sub    $0x10,%rsp
   0x000000000040052e <+8>:     movl   $0x0,-0x8(%rbp)
   0x0000000000400535 <+15>:    movl   $0x0,-0xc(%rbp)
   0x000000000040053c <+22>:    jmp    0x400548 <main+34>
   0x000000000040053e <+24>:    mov    -0xc(%rbp),%eax
   0x0000000000400541 <+27>:    add    %eax,-0x8(%rbp)
   0x0000000000400544 <+30>:    addl   $0x1,-0xc(%rbp)
   0x0000000000400548 <+34>:    cmpl   $0x4,-0xc(%rbp)
   0x000000000040054c <+38>:    jle    0x40053e <main+24>
   0x000000000040054e <+40>:    mov    -0x8(%rbp),%eax
   0x0000000000400551 <+43>:    mov    %eax,%edi
   0x0000000000400553 <+45>:    callq  0x400576 <multi2>
   0x0000000000400558 <+50>:    mov    %eax,-0x4(%rbp)
   0x000000000040055b <+53>:    mov    -0x4(%rbp),%eax
   0x000000000040055e <+56>:    mov    %eax,%esi
   0x0000000000400560 <+58>:    mov    $0x400614,%edi
   0x0000000000400565 <+63>:    mov    $0x0,%eax
   0x000000000040056a <+68>:    callq  0x400400 <printf@plt>
   0x000000000040056f <+73>:    mov    $0x0,%eax
   0x0000000000400574 <+78>:    leaveq
   0x0000000000400575 <+79>:    retq
End of assembler dump.
```

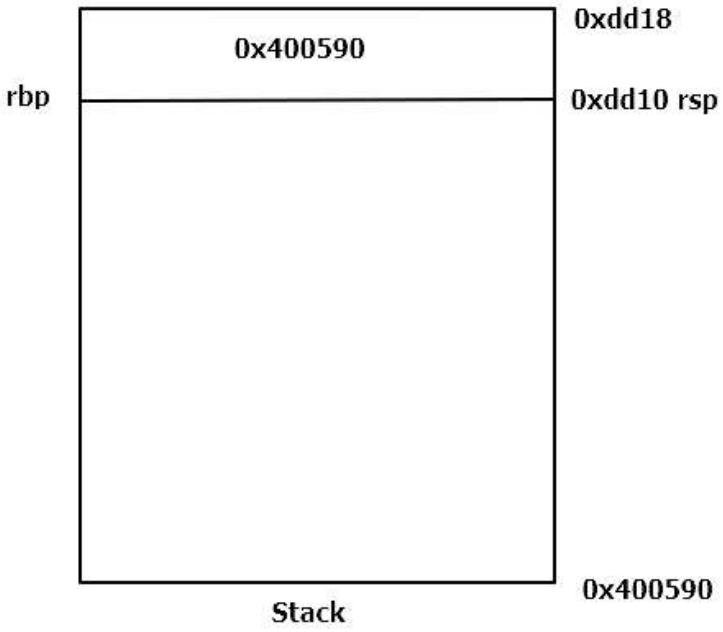


Stack

rsp	0xdd10
rbp	0x400590
rip	0x400527 (mov %rsp,%rbp)
eax	0x400526
edi	0x1

Registers in CPU

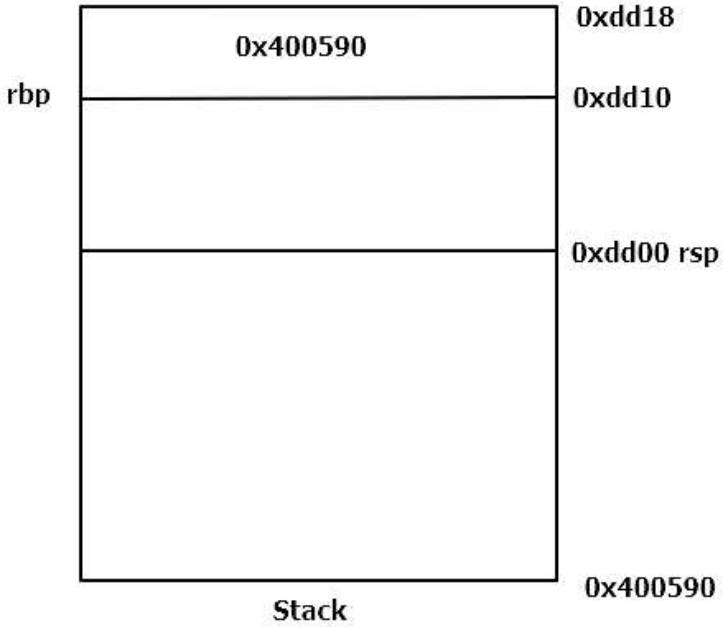
```
Dump of assembler code for function main:
   0x0000000000400526 <+0>:     push   %rbp
=> 0x0000000000400527 <+1>:     mov    %rsp,%rbp
   0x000000000040052a <+4>:     sub    $0x10,%rsp
   0x000000000040052e <+8>:     movl   $0x0,-0x8(%rbp)
   0x0000000000400535 <+15>:    movl   $0x0,-0xc(%rbp)
   0x000000000040053c <+22>:    jmp    0x400548 <main+34>
   0x000000000040053e <+24>:    mov    -0xc(%rbp),%eax
   0x0000000000400541 <+27>:    add    %eax,-0x8(%rbp)
   0x0000000000400544 <+30>:    addl   $0x1,-0xc(%rbp)
   0x0000000000400548 <+34>:    cmpl   $0x4,-0xc(%rbp)
   0x000000000040054c <+38>:    jle    0x40053e <main+24>
   0x000000000040054e <+40>:    mov    -0x8(%rbp),%eax
   0x0000000000400551 <+43>:    mov    %eax,%edi
   0x0000000000400553 <+45>:    callq  0x400576 <multi2>
   0x0000000000400558 <+50>:    mov    %eax,-0x4(%rbp)
   0x000000000040055b <+53>:    mov    -0x4(%rbp),%eax
   0x000000000040055e <+56>:    mov    %eax,%esi
   0x0000000000400560 <+58>:    mov    $0x400614,%edi
   0x0000000000400565 <+63>:    mov    $0x0,%eax
   0x000000000040056a <+68>:    callq  0x400400 <printf@plt>
   0x000000000040056f <+73>:    mov    $0x0,%eax
   0x0000000000400574 <+78>:    leaveq
   0x0000000000400575 <+79>:    retq
End of assembler dump.
```



rsp	0xdd10
rbp	0xdd10
rip	0x40052a (sub $0x10,%rsp)
eax	0x400526
edi	0x1

Registers in CPU

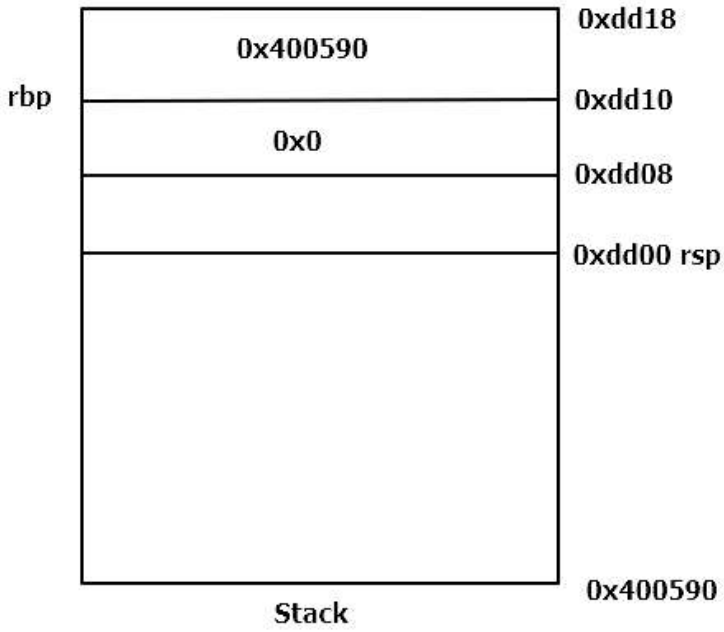
```
Dump of assembler code for function main:
   0x0000000000400526 <+0>:     push   %rbp
   0x0000000000400527 <+1>:     mov    %rsp,%rbp
=> 0x000000000040052a <+4>:     sub    $0x10,%rsp
   0x000000000040052e <+8>:     movl   $0x0,-0x8(%rbp)
   0x0000000000400535 <+15>:    movl   $0x0,-0xc(%rbp)
   0x000000000040053c <+22>:    jmp    0x400548 <main+34>
   0x000000000040053e <+24>:    mov    -0xc(%rbp),%eax
   0x0000000000400541 <+27>:    add    %eax,-0x8(%rbp)
   0x0000000000400544 <+30>:    addl   $0x1,-0xc(%rbp)
   0x0000000000400548 <+34>:    cmpl   $0x4,-0xc(%rbp)
   0x000000000040054c <+38>:    jle    0x40053e <main+24>
   0x000000000040054e <+40>:    mov    -0x8(%rbp),%eax
   0x0000000000400551 <+43>:    mov    %eax,%edi
   0x0000000000400553 <+45>:    callq  0x400576 <multi2>
   0x0000000000400558 <+50>:    mov    %eax,-0x4(%rbp)
   0x000000000040055b <+53>:    mov    -0x4(%rbp),%eax
   0x000000000040055e <+56>:    mov    %eax,%esi
   0x0000000000400560 <+58>:    mov    $0x400614,%edi
   0x0000000000400565 <+63>:    mov    $0x0,%eax
   0x000000000040056a <+68>:    callq  0x400400 <printf@plt>
   0x000000000040056f <+73>:    mov    $0x0,%eax
   0x0000000000400574 <+78>:    leaveq
   0x0000000000400575 <+79>:    retq
End of assembler dump.
```



rsp	0xdd10
rbp	0xdd10
rip	0x40052e (movl $0x0, -0x8(%rbp))
eax	0x400526
edi	0x1

Registers in CPU

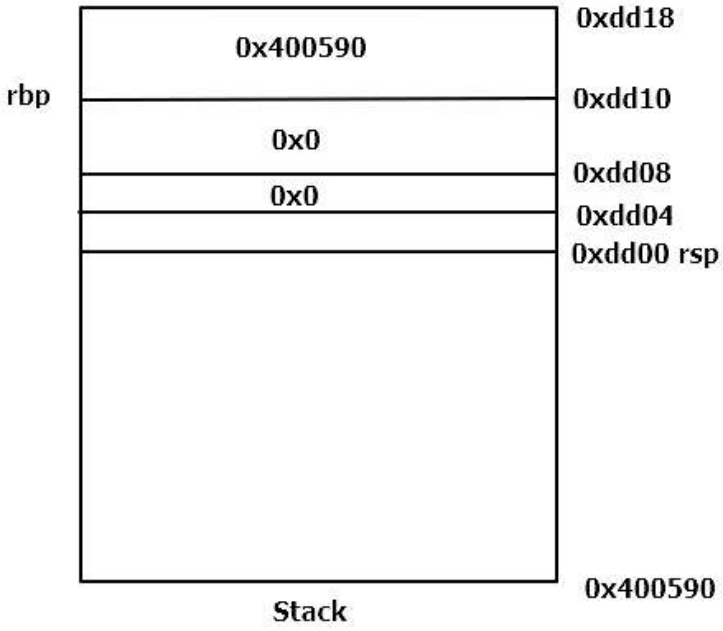
```
Dump of assembler code for function main:
   0x0000000000400526 <+0>:     push   %rbp
   0x0000000000400527 <+1>:     mov    %rsp,%rbp
   0x000000000040052a <+4>:     sub    $0x10,%rsp
=> 0x000000000040052e <+8>:     movl   $0x0,-0x8(%rbp)
   0x0000000000400535 <+15>:    movl   $0x0,-0xc(%rbp)
   0x000000000040053c <+22>:    jmp    0x400548 <main+34>
   0x000000000040053e <+24>:    mov    -0xc(%rbp),%eax
   0x0000000000400541 <+27>:    add    %eax,-0x8(%rbp)
   0x0000000000400544 <+30>:    addl   $0x1,-0xc(%rbp)
   0x0000000000400548 <+34>:    cmpl   $0x4,-0xc(%rbp)
   0x000000000040054c <+38>:    jle    0x40053e <main+24>
   0x000000000040054e <+40>:    mov    -0x8(%rbp),%eax
   0x0000000000400551 <+43>:    mov    %eax,%edi
   0x0000000000400553 <+45>:    callq  0x400576 <multi2>
   0x0000000000400558 <+50>:    mov    %eax,-0x4(%rbp)
   0x000000000040055b <+53>:    mov    -0x4(%rbp),%eax
   0x000000000040055e <+56>:    mov    %eax,%esi
   0x0000000000400560 <+58>:    mov    $0x400614,%edi
   0x0000000000400565 <+63>:    mov    $0x0,%eax
   0x000000000040056a <+68>:    callq  0x400400 <printf@plt>
   0x000000000040056f <+73>:    mov    $0x0,%eax
   0x0000000000400574 <+78>:    leaveq
   0x0000000000400575 <+79>:    retq
End of assembler dump.
```



rsp	0xdd10
rbp	0xdd10
rip	0x400535 (movl $0x0, -0xc(%rbp))
eax	0x400526
edi	0x1

Registers in CPU

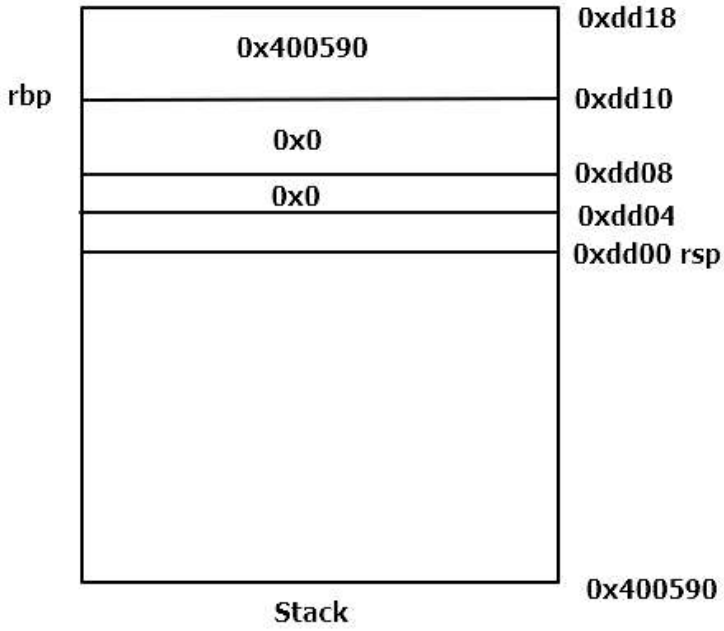
```
Dump of assembler code for function main:
   0x0000000000400526 <+0>:     push   %rbp
   0x0000000000400527 <+1>:     mov    %rsp,%rbp
   0x000000000040052a <+4>:     sub    $0x10,%rsp
   0x000000000040052e <+8>:     movl   $0x0,-0x8(%rbp)
=> 0x0000000000400535 <+15>:    movl   $0x0,-0xc(%rbp)
   0x000000000040053c <+22>:    jmp    0x400548 <main+34>
   0x000000000040053e <+24>:    mov    -0xc(%rbp),%eax
   0x0000000000400541 <+27>:    add    %eax,-0x8(%rbp)
   0x0000000000400544 <+30>:    addl   $0x1,-0xc(%rbp)
   0x0000000000400548 <+34>:    cmpl   $0x4,-0xc(%rbp)
   0x000000000040054c <+38>:    jle    0x40053e <main+24>
   0x000000000040054e <+40>:    mov    -0x8(%rbp),%eax
   0x0000000000400551 <+43>:    mov    %eax,%edi
   0x0000000000400553 <+45>:    callq  0x400576 <multi2>
   0x0000000000400558 <+50>:    mov    %eax,-0x4(%rbp)
   0x000000000040055b <+53>:    mov    -0x4(%rbp),%eax
   0x000000000040055e <+56>:    mov    %eax,%esi
   0x0000000000400560 <+58>:    mov    $0x400614,%edi
   0x0000000000400565 <+63>:    mov    $0x0,%eax
   0x000000000040056a <+68>:    callq  0x400400 <printf@plt>
   0x000000000040056f <+73>:    mov    $0x0,%eax
   0x0000000000400574 <+78>:    leaveq
   0x0000000000400575 <+79>:    retq
End of assembler dump.
```



rsp	0xdd10
rbp	0xdd10
rip	0x40054c (jmp 0x400548 <main+34>)
eax	0x400526
edi	0x1

Registers in CPU

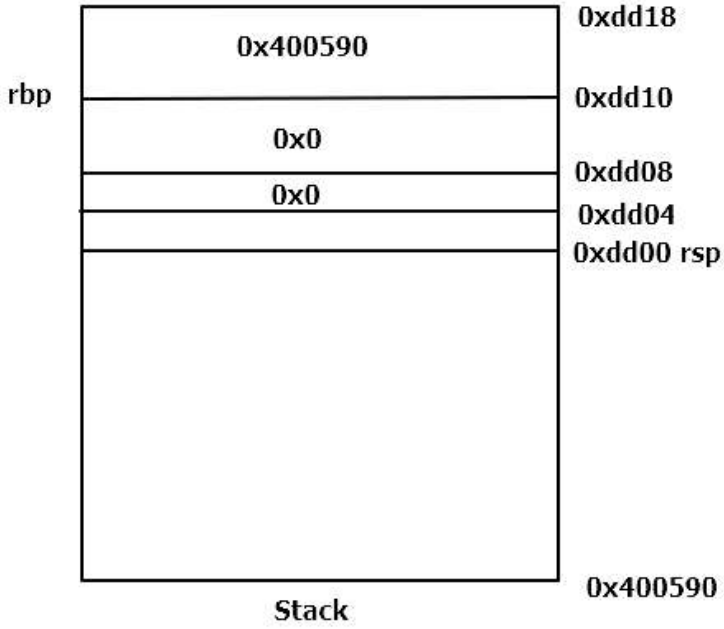
```
Dump of assembler code for function main:
   0x0000000000400526 <+0>:     push   %rbp
   0x0000000000400527 <+1>:     mov    %rsp,%rbp
   0x000000000040052a <+4>:     sub    $0x10,%rsp
   0x000000000040052e <+8>:     movl   $0x0,-0x8(%rbp)
   0x0000000000400535 <+15>:    movl   $0x0,-0xc(%rbp)
=> 0x000000000040053c <+22>:    jmp    0x400548 <main+34>
   0x000000000040053e <+24>:    mov    -0xc(%rbp),%eax
   0x0000000000400541 <+27>:    add    %eax,-0x8(%rbp)
   0x0000000000400544 <+30>:    addl   $0x1,-0xc(%rbp)
   0x0000000000400548 <+34>:    cmpl   $0x4,-0xc(%rbp)
   0x000000000040054c <+38>:    jle    0x40053e <main+24>
   0x000000000040054e <+40>:    mov    -0x8(%rbp),%eax
   0x0000000000400551 <+43>:    mov    %eax,%edi
   0x0000000000400553 <+45>:    callq  0x400576 <multi2>
   0x0000000000400558 <+50>:    mov    %eax,-0x4(%rbp)
   0x000000000040055b <+53>:    mov    -0x4(%rbp),%eax
   0x000000000040055e <+56>:    mov    %eax,%esi
   0x0000000000400560 <+58>:    mov    $0x400614,%edi
   0x0000000000400565 <+63>:    mov    $0x0,%eax
   0x000000000040056a <+68>:    callq  0x400400 <printf@plt>
   0x000000000040056f <+73>:    mov    $0x0,%eax
   0x0000000000400574 <+78>:    leaveq
   0x0000000000400575 <+79>:    retq
End of assembler dump.
```



rsp	0xdd10
rbp	0xdd10
rip	0x400548 (cmpl $0x4,-0xc(%rbp))
eax	0x400526
edi	0x1

Registers in CPU

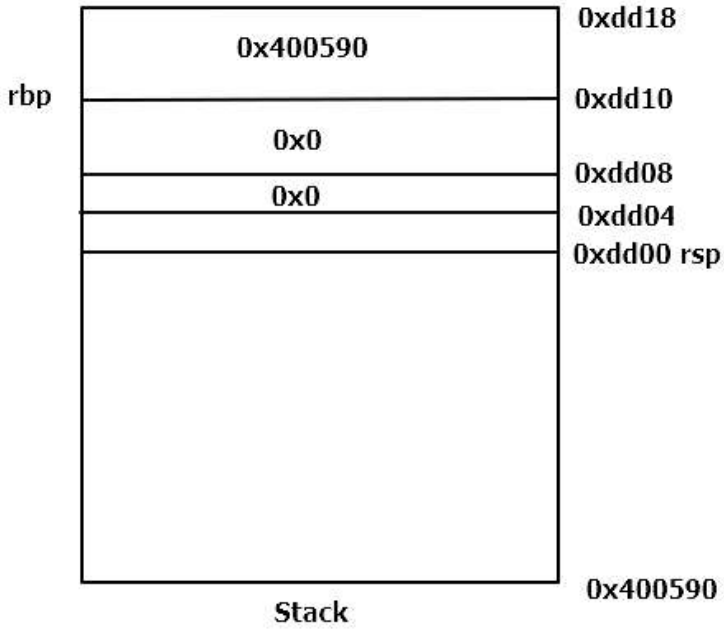
```
Dump of assembler code for function main:
   0x0000000000400526 <+0>:     push   %rbp
   0x0000000000400527 <+1>:     mov    %rsp,%rbp
   0x000000000040052a <+4>:     sub    $0x10,%rsp
   0x000000000040052e <+8>:     movl   $0x0,-0x8(%rbp)
   0x0000000000400535 <+15>:    movl   $0x0,-0xc(%rbp)
   0x000000000040053c <+22>:    jmp    0x400548 <main+34>
   0x000000000040053e <+24>:    mov    -0xc(%rbp),%eax
   0x0000000000400541 <+27>:    add    %eax,-0x8(%rbp)
   0x0000000000400544 <+30>:    addl   $0x1,-0xc(%rbp)
=> 0x0000000000400548 <+34>:    cmpl   $0x4,-0xc(%rbp)
   0x000000000040054c <+38>:    jle    0x40053e <main+24>
   0x000000000040054e <+40>:    mov    -0x8(%rbp),%eax
   0x0000000000400551 <+43>:    mov    %eax,%edi
   0x0000000000400553 <+45>:    callq  0x400576 <multi2>
   0x0000000000400558 <+50>:    mov    %eax,-0x4(%rbp)
   0x000000000040055b <+53>:    mov    -0x4(%rbp),%eax
   0x000000000040055e <+56>:    mov    %eax,%esi
   0x0000000000400560 <+58>:    mov    $0x400614,%edi
   0x0000000000400565 <+63>:    mov    $0x0,%eax
   0x000000000040056a <+68>:    callq  0x400400 <printf@plt>
   0x000000000040056f <+73>:    mov    $0x0,%eax
   0x0000000000400574 <+78>:    leaveq
   0x0000000000400575 <+79>:    retq
End of assembler dump.
```



rsp	0xdd10
rbp	0xdd10
rip	0x40054c (jle 0x40053e <main+24>)
eax	0x400526
edi	0x1

Registers in CPU

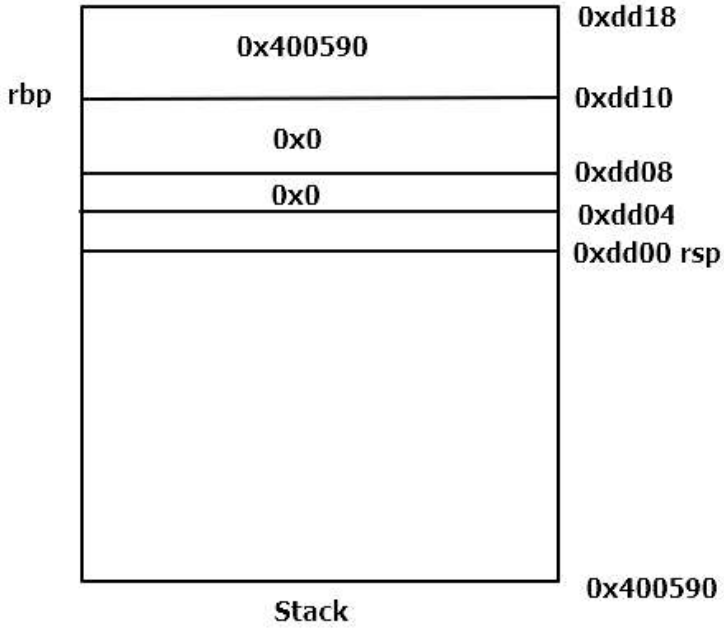
```
Dump of assembler code for function main:
   0x0000000000400526 <+0>:     push   %rbp
   0x0000000000400527 <+1>:     mov    %rsp,%rbp
   0x000000000040052a <+4>:     sub    $0x10,%rsp
   0x000000000040052e <+8>:     movl   $0x0,-0x8(%rbp)
   0x0000000000400535 <+15>:    movl   $0x0,-0xc(%rbp)
   0x000000000040053c <+22>:    jmp    0x400548 <main+34>
   0x000000000040053e <+24>:    mov    -0xc(%rbp),%eax
   0x0000000000400541 <+27>:    add    %eax,-0x8(%rbp)
   0x0000000000400544 <+30>:    addl   $0x1,-0xc(%rbp)
   0x0000000000400548 <+34>:    cmpl   $0x4,-0xc(%rbp)
=> 0x000000000040054c <+38>:    jle    0x40053e <main+24>
   0x000000000040054e <+40>:    mov    -0x8(%rbp),%eax
   0x0000000000400551 <+43>:    mov    %eax,%edi
   0x0000000000400553 <+45>:    callq  0x400576 <multi2>
   0x0000000000400558 <+50>:    mov    %eax,-0x4(%rbp)
   0x000000000040055b <+53>:    mov    -0x4(%rbp),%eax
   0x000000000040055e <+56>:    mov    %eax,%esi
   0x0000000000400560 <+58>:    mov    $0x400614,%edi
   0x0000000000400565 <+63>:    mov    $0x0,%eax
   0x000000000040056a <+68>:    callq  0x400400 <printf@plt>
   0x000000000040056f <+73>:    mov    $0x0,%eax
   0x0000000000400574 <+78>:    leaveq
   0x0000000000400575 <+79>:    retq
End of assembler dump.
```



rsp	0xdd10
rbp	0xdd10
rip	0x40053e (mov -0xc(%rbp),%eax)
eax	0x400526
edi	0x1

Registers in CPU

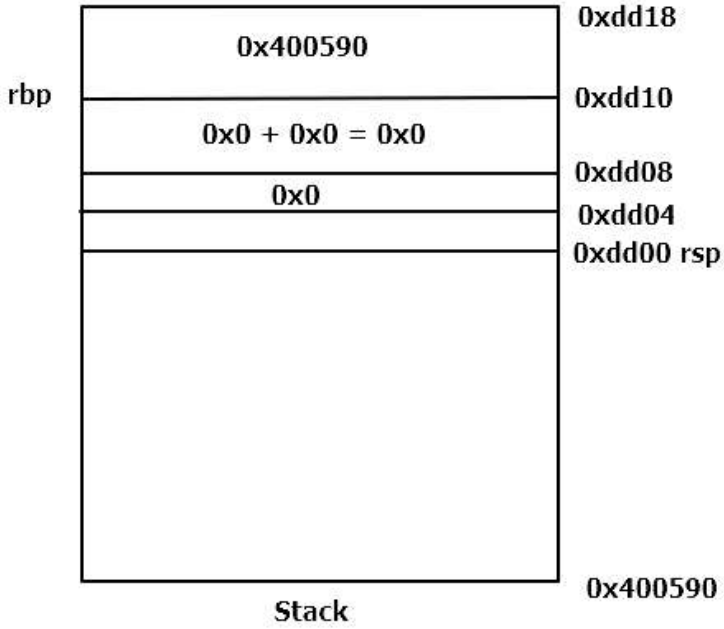
```
Dump of assembler code for function main:
   0x0000000000400526 <+0>:     push   %rbp
   0x0000000000400527 <+1>:     mov    %rsp,%rbp
   0x000000000040052a <+4>:     sub    $0x10,%rsp
   0x000000000040052e <+8>:     movl   $0x0,-0x8(%rbp)
   0x0000000000400535 <+15>:    movl   $0x0,-0xc(%rbp)
   0x000000000040053c <+22>:    jmp    0x400548 <main+34>
=> 0x000000000040053e <+24>:    mov    -0xc(%rbp),%eax
   0x0000000000400541 <+27>:    add    %eax,-0x8(%rbp)
   0x0000000000400544 <+30>:    addl   $0x1,-0xc(%rbp)
   0x0000000000400548 <+34>:    cmpl   $0x4,-0xc(%rbp)
   0x000000000040054c <+38>:    jle    0x40053e <main+24>
   0x000000000040054e <+40>:    mov    -0x8(%rbp),%eax
   0x0000000000400551 <+43>:    mov    %eax,%edi
   0x0000000000400553 <+45>:    callq  0x400576 <multi2>
   0x0000000000400558 <+50>:    mov    %eax,-0x4(%rbp)
   0x000000000040055b <+53>:    mov    -0x4(%rbp),%eax
   0x000000000040055e <+56>:    mov    %eax,%esi
   0x0000000000400560 <+58>:    mov    $0x400614,%edi
   0x0000000000400565 <+63>:    mov    $0x0,%eax
   0x000000000040056a <+68>:    callq  0x400400 <printf@plt>
   0x000000000040056f <+73>:    mov    $0x0,%eax
   0x0000000000400574 <+78>:    leaveq
   0x0000000000400575 <+79>:    retq
End of assembler dump.
```



rsp	0xdd10
rbp	0xdd10
rip	0x400541 (add %eax,-0x8(%rbp))
eax	0x0
edi	0x1

Registers in CPU

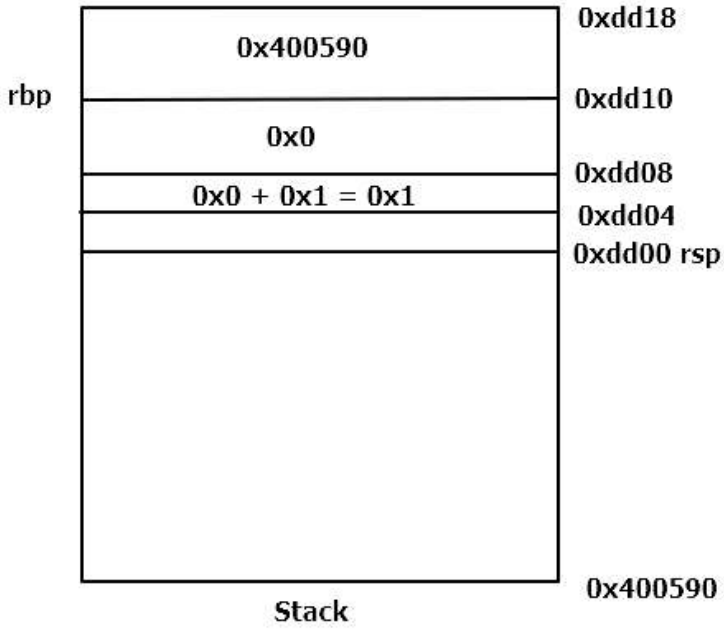
```
Dump of assembler code for function main:
   0x0000000000400526 <+0>:     push   %rbp
   0x0000000000400527 <+1>:     mov    %rsp,%rbp
   0x000000000040052a <+4>:     sub    $0x10,%rsp
   0x000000000040052e <+8>:     movl   $0x0,-0x8(%rbp)
   0x0000000000400535 <+15>:    movl   $0x0,-0xc(%rbp)
   0x000000000040053c <+22>:    jmp    0x400548 <main+34>
   0x000000000040053e <+24>:    mov    -0xc(%rbp),%eax
=> 0x0000000000400541 <+27>:    add    %eax,-0x8(%rbp)
   0x0000000000400544 <+30>:    addl   $0x1,-0xc(%rbp)
   0x0000000000400548 <+34>:    cmpl   $0x4,-0xc(%rbp)
   0x000000000040054c <+38>:    jle    0x40053e <main+24>
   0x000000000040054e <+40>:    mov    -0x8(%rbp),%eax
   0x0000000000400551 <+43>:    mov    %eax,%edi
   0x0000000000400553 <+45>:    callq  0x400576 <multi2>
   0x0000000000400558 <+50>:    mov    %eax,-0x4(%rbp)
   0x000000000040055b <+53>:    mov    -0x4(%rbp),%eax
   0x000000000040055e <+56>:    mov    %eax,%esi
   0x0000000000400560 <+58>:    mov    $0x400614,%edi
   0x0000000000400565 <+63>:    mov    $0x0,%eax
   0x000000000040056a <+68>:    callq  0x400400 <printf@plt>
   0x000000000040056f <+73>:    mov    $0x0,%eax
   0x0000000000400574 <+78>:    leaveq
   0x0000000000400575 <+79>:    retq
End of assembler dump.
```



rsp	0xdd10
rbp	0xdd10
rip	0x400544 (addl &0x1,-0xc(%rbp))
eax	0x0
edi	0x1

Registers in CPU

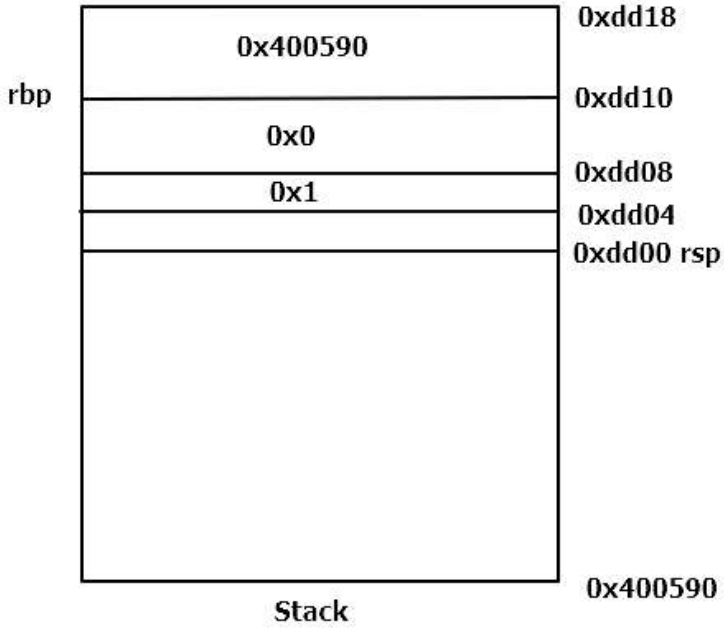
```
Dump of assembler code for function main:
   0x0000000000400526 <+0>:     push   %rbp
   0x0000000000400527 <+1>:     mov    %rsp,%rbp
   0x000000000040052a <+4>:     sub    $0x10,%rsp
   0x000000000040052e <+8>:     movl   $0x0,-0x8(%rbp)
   0x0000000000400535 <+15>:    movl   $0x0,-0xc(%rbp)
   0x000000000040053c <+22>:    jmp    0x400548 <main+34>
   0x000000000040053e <+24>:    mov    -0xc(%rbp),%eax
   0x0000000000400541 <+27>:    add    %eax,-0x8(%rbp)
=> 0x0000000000400544 <+30>:    addl   $0x1,-0xc(%rbp)
   0x0000000000400548 <+34>:    cmpl   $0x4,-0xc(%rbp)
   0x000000000040054c <+38>:    jle    0x40053e <main+24>
   0x000000000040054e <+40>:    mov    -0x8(%rbp),%eax
   0x0000000000400551 <+43>:    mov    %eax,%edi
   0x0000000000400553 <+45>:    callq  0x400576 <multi2>
   0x0000000000400558 <+50>:    mov    %eax,-0x4(%rbp)
   0x000000000040055b <+53>:    mov    -0x4(%rbp),%eax
   0x000000000040055e <+56>:    mov    %eax,%esi
   0x0000000000400560 <+58>:    mov    $0x400614,%edi
   0x0000000000400565 <+63>:    mov    $0x0,%eax
   0x000000000040056a <+68>:    callq  0x400400 <printf@plt>
   0x000000000040056f <+73>:    mov    $0x0,%eax
   0x0000000000400574 <+78>:    leaveq
   0x0000000000400575 <+79>:    retq
End of assembler dump.
```



rsp	0xdd10
rbp	0xdd10
rip	0x400548 (cmpl $0x4,-0xc(%rbp)
eax	0x0
edi	0x1

Registers in CPU

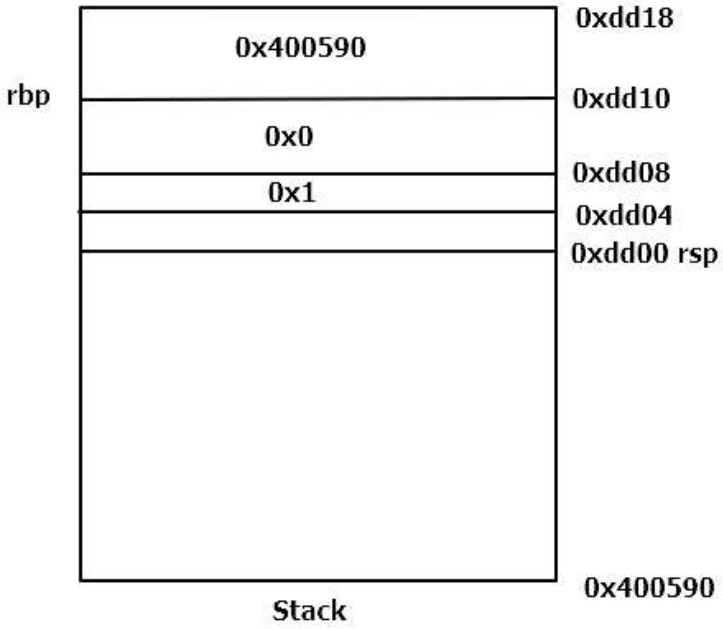
```
Dump of assembler code for function main:
   0x0000000000400526 <+0>:     push   %rbp
   0x0000000000400527 <+1>:     mov    %rsp,%rbp
   0x000000000040052a <+4>:     sub    $0x10,%rsp
   0x000000000040052e <+8>:     movl   $0x0,-0x8(%rbp)
   0x0000000000400535 <+15>:    movl   $0x0,-0xc(%rbp)
   0x000000000040053c <+22>:    jmp    0x400548 <main+34>
   0x000000000040053e <+24>:    mov    -0xc(%rbp),%eax
   0x0000000000400541 <+27>:    add    %eax,-0x8(%rbp)
   0x0000000000400544 <+30>:    addl   $0x1,-0xc(%rbp)
=> 0x0000000000400548 <+34>:    cmpl   $0x4,-0xc(%rbp)
   0x000000000040054c <+38>:    jle    0x40053e <main+24>
   0x000000000040054e <+40>:    mov    -0x8(%rbp),%eax
   0x0000000000400551 <+43>:    mov    %eax,%edi
   0x0000000000400553 <+45>:    callq  0x400576 <multi2>
   0x0000000000400558 <+50>:    mov    %eax,-0x4(%rbp)
   0x000000000040055b <+53>:    mov    -0x4(%rbp),%eax
   0x000000000040055e <+56>:    mov    %eax,%esi
   0x0000000000400560 <+58>:    mov    $0x400614,%edi
   0x0000000000400565 <+63>:    mov    $0x0,%eax
   0x000000000040056a <+68>:    callq  0x400400 <printf@plt>
   0x000000000040056f <+73>:    mov    $0x0,%eax
   0x0000000000400574 <+78>:    leaveq
   0x0000000000400575 <+79>:    retq
End of assembler dump.
```



rsp	0xdd10
rbp	0xdd10
rip	0x40054c (jle 0x40053e <main+24>
eax	0x0
edi	0x1

Registers in CPU

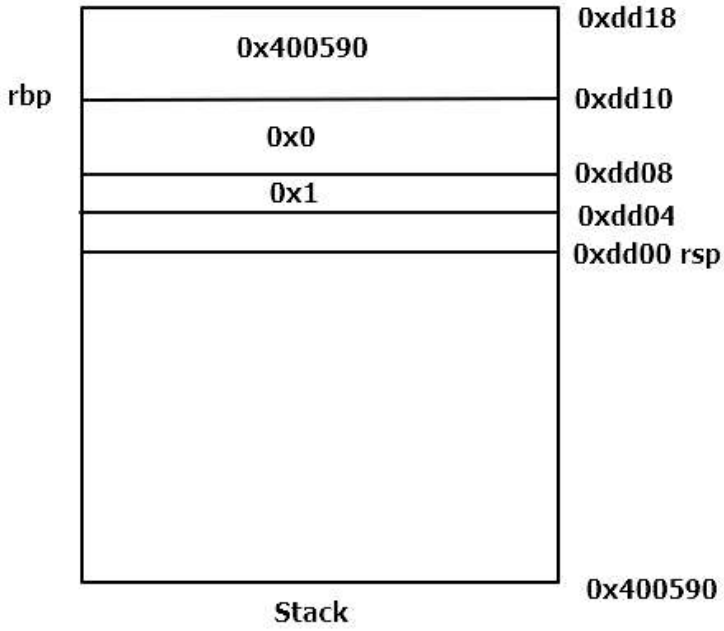
```
Dump of assembler code for function main:
   0x0000000000400526 <+0>:     push   %rbp
   0x0000000000400527 <+1>:     mov    %rsp,%rbp
   0x000000000040052a <+4>:     sub    $0x10,%rsp
   0x000000000040052e <+8>:     movl   $0x0,-0x8(%rbp)
   0x0000000000400535 <+15>:    movl   $0x0,-0xc(%rbp)
   0x000000000040053c <+22>:    jmp    0x400548 <main+34>
   0x000000000040053e <+24>:    mov    -0xc(%rbp),%eax
   0x0000000000400541 <+27>:    add    %eax,-0x8(%rbp)
   0x0000000000400544 <+30>:    addl   $0x1,-0xc(%rbp)
   0x0000000000400548 <+34>:    cmpl   $0x4,-0xc(%rbp)
=> 0x000000000040054c <+38>:    jle    0x40053e <main+24>
   0x000000000040054e <+40>:    mov    -0x8(%rbp),%eax
   0x0000000000400551 <+43>:    mov    %eax,%edi
   0x0000000000400553 <+45>:    callq  0x400576 <multi2>
   0x0000000000400558 <+50>:    mov    %eax,-0x4(%rbp)
   0x000000000040055b <+53>:    mov    -0x4(%rbp),%eax
   0x000000000040055e <+56>:    mov    %eax,%esi
   0x0000000000400560 <+58>:    mov    $0x400614,%edi
   0x0000000000400565 <+63>:    mov    $0x0,%eax
   0x000000000040056a <+68>:    callq  0x400400 <printf@plt>
   0x000000000040056f <+73>:    mov    $0x0,%eax
   0x0000000000400574 <+78>:    leaveq
   0x0000000000400575 <+79>:    retq
End of assembler dump.
```



rsp	0xdd10
rbp	0xdd10
rip	0x40053e (mov -0xc(%rbp),%eax)
eax	0x0
edi	0x1

Registers in CPU

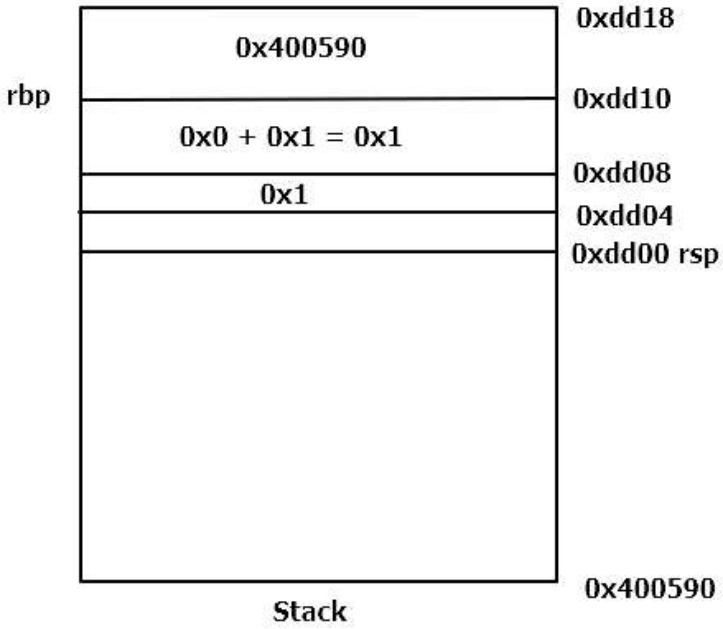
```
Dump of assembler code for function main:
   0x0000000000400526 <+0>:     push   %rbp
   0x0000000000400527 <+1>:     mov    %rsp,%rbp
   0x000000000040052a <+4>:     sub    $0x10,%rsp
   0x000000000040052e <+8>:     movl   $0x0,-0x8(%rbp)
   0x0000000000400535 <+15>:    movl   $0x0,-0xc(%rbp)
   0x000000000040053c <+22>:    jmp    0x400548 <main+34>
=> 0x000000000040053e <+24>:    mov    -0xc(%rbp),%eax
   0x0000000000400541 <+27>:    add    %eax,-0x8(%rbp)
   0x0000000000400544 <+30>:    addl   $0x1,-0xc(%rbp)
   0x0000000000400548 <+34>:    cmpl   $0x4,-0xc(%rbp)
   0x000000000040054c <+38>:    jle    0x40053e <main+24>
   0x000000000040054e <+40>:    mov    -0x8(%rbp),%eax
   0x0000000000400551 <+43>:    mov    %eax,%edi
   0x0000000000400553 <+45>:    callq  0x400576 <multi2>
   0x0000000000400558 <+50>:    mov    %eax,-0x4(%rbp)
   0x000000000040055b <+53>:    mov    -0x4(%rbp),%eax
   0x000000000040055e <+56>:    mov    %eax,%esi
   0x0000000000400560 <+58>:    mov    $0x400614,%edi
   0x0000000000400565 <+63>:    mov    $0x0,%eax
   0x000000000040056a <+68>:    callq  0x400400 <printf@plt>
   0x000000000040056f <+73>:    mov    $0x0,%eax
   0x0000000000400574 <+78>:    leaveq
   0x0000000000400575 <+79>:    retq
End of assembler dump.
```



rsp	0xdd10
rbp	0xdd10
rip	0x400541 (add %eax,-0x8(%rbp))
eax	0x1
edi	0x1

Registers in CPU

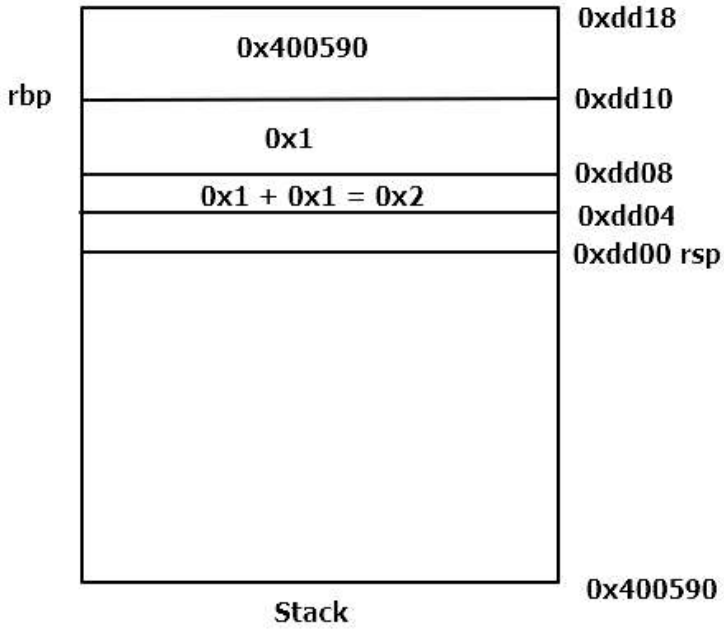
```
Dump of assembler code for function main:
   0x0000000000400526 <+0>:     push   %rbp
   0x0000000000400527 <+1>:     mov    %rsp,%rbp
   0x000000000040052a <+4>:     sub    $0x10,%rsp
   0x000000000040052e <+8>:     movl   $0x0,-0x8(%rbp)
   0x0000000000400535 <+15>:    movl   $0x0,-0xc(%rbp)
   0x000000000040053c <+22>:    jmp    0x400548 <main+34>
   0x000000000040053e <+24>:    mov    -0xc(%rbp),%eax
=> 0x0000000000400541 <+27>:    add    %eax,-0x8(%rbp)
   0x0000000000400544 <+30>:    addl   $0x1,-0xc(%rbp)
   0x0000000000400548 <+34>:    cmpl   $0x4,-0xc(%rbp)
   0x000000000040054c <+38>:    jle    0x40053e <main+24>
   0x000000000040054e <+40>:    mov    -0x8(%rbp),%eax
   0x0000000000400551 <+43>:    mov    %eax,%edi
   0x0000000000400553 <+45>:    callq  0x400576 <multi2>
   0x0000000000400558 <+50>:    mov    %eax,-0x4(%rbp)
   0x000000000040055b <+53>:    mov    -0x4(%rbp),%eax
   0x000000000040055e <+56>:    mov    %eax,%esi
   0x0000000000400560 <+58>:    mov    $0x400614,%edi
   0x0000000000400565 <+63>:    mov    $0x0,%eax
   0x000000000040056a <+68>:    callq  0x400400 <printf@plt>
   0x000000000040056f <+73>:    mov    $0x0,%eax
   0x0000000000400574 <+78>:    leaveq
   0x0000000000400575 <+79>:    retq
End of assembler dump.
```



rsp	0xdd10
rbp	0xdd10
rip	400544 (addl $0x1,-0xc(%rbp))
eax	0x1
edi	0x1

Registers in CPU

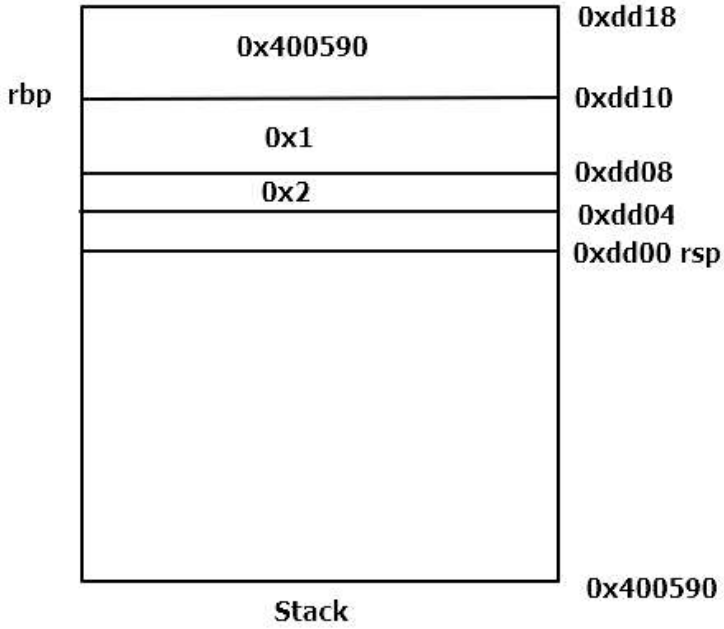
```
Dump of assembler code for function main:
   0x0000000000400526 <+0>:     push   %rbp
   0x0000000000400527 <+1>:     mov    %rsp,%rbp
   0x000000000040052a <+4>:     sub    $0x10,%rsp
   0x000000000040052e <+8>:     movl   $0x0,-0x8(%rbp)
   0x0000000000400535 <+15>:    movl   $0x0,-0xc(%rbp)
   0x000000000040053c <+22>:    jmp    0x400548 <main+34>
   0x000000000040053e <+24>:    mov    -0xc(%rbp),%eax
   0x0000000000400541 <+27>:    add    %eax,-0x8(%rbp)
=> 0x0000000000400544 <+30>:    addl   $0x1,-0xc(%rbp)
   0x0000000000400548 <+34>:    cmpl   $0x4,-0xc(%rbp)
   0x000000000040054c <+38>:    jle    0x40053e <main+24>
   0x000000000040054e <+40>:    mov    -0x8(%rbp),%eax
   0x0000000000400551 <+43>:    mov    %eax,%edi
   0x0000000000400553 <+45>:    callq  0x400576 <multi2>
   0x0000000000400558 <+50>:    mov    %eax,-0x4(%rbp)
   0x000000000040055b <+53>:    mov    -0x4(%rbp),%eax
   0x000000000040055e <+56>:    mov    %eax,%esi
   0x0000000000400560 <+58>:    mov    $0x400614,%edi
   0x0000000000400565 <+63>:    mov    $0x0,%eax
   0x000000000040056a <+68>:    callq  0x400400 <printf@plt>
   0x000000000040056f <+73>:    mov    $0x0,%eax
   0x0000000000400574 <+78>:    leaveq
   0x0000000000400575 <+79>:    retq
End of assembler dump.
```



rsp	0xdd10
rbp	0xdd10
rip	0x400548 (cmpl &0x4, -0xc(%rbp)
eax	0x1
edi	0x1

Registers in CPU

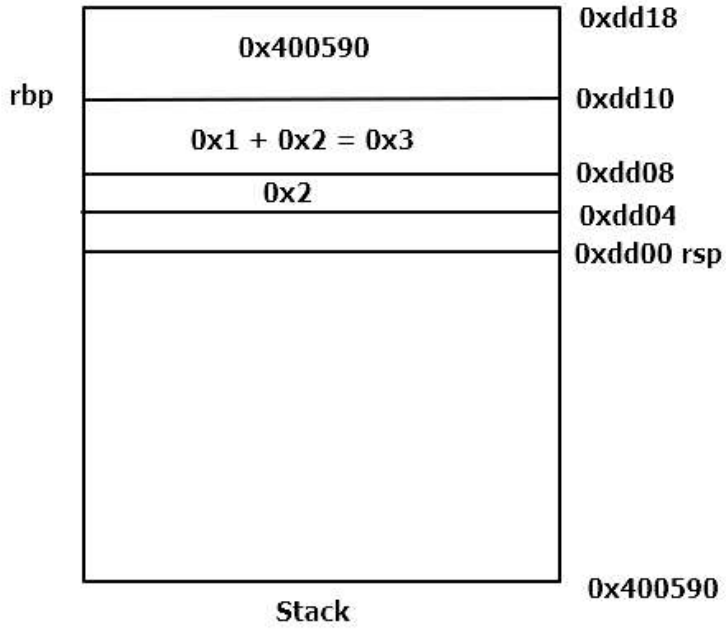
```
Dump of assembler code for function main:
   0x0000000000400526 <+0>:     push   %rbp
   0x0000000000400527 <+1>:     mov    %rsp,%rbp
   0x000000000040052a <+4>:     sub    $0x10,%rsp
   0x000000000040052e <+8>:     movl   $0x0,-0x8(%rbp)
   0x0000000000400535 <+15>:    movl   $0x0,-0xc(%rbp)
   0x000000000040053c <+22>:    jmp    0x400548 <main+34>
   0x000000000040053e <+24>:    mov    -0xc(%rbp),%eax
   0x0000000000400541 <+27>:    add    %eax,-0x8(%rbp)
   0x0000000000400544 <+30>:    addl   $0x1,-0xc(%rbp)
=> 0x0000000000400548 <+34>:    cmpl   $0x4,-0xc(%rbp)
   0x000000000040054c <+38>:    jle    0x40053e <main+24>
   0x000000000040054e <+40>:    mov    -0x8(%rbp),%eax
   0x0000000000400551 <+43>:    mov    %eax,%edi
   0x0000000000400553 <+45>:    callq  0x400576 <multi2>
   0x0000000000400558 <+50>:    mov    %eax,-0x4(%rbp)
   0x000000000040055b <+53>:    mov    -0x4(%rbp),%eax
   0x000000000040055e <+56>:    mov    %eax,%esi
   0x0000000000400560 <+58>:    mov    $0x400614,%edi
   0x0000000000400565 <+63>:    mov    $0x0,%eax
   0x000000000040056a <+68>:    callq  0x400400 <printf@plt>
   0x000000000040056f <+73>:    mov    $0x0,%eax
   0x0000000000400574 <+78>:    leaveq
   0x0000000000400575 <+79>:    retq
End of assembler dump.
```



rsp	0xdd10
rbp	0xdd10
rip	0x400541 (add %eax,-0x8(%rbp))
eax	0x2
edi	0x1

Registers in CPU

```
Dump of assembler code for function main:
   0x0000000000400526 <+0>:     push   %rbp
   0x0000000000400527 <+1>:     mov    %rsp,%rbp
   0x000000000040052a <+4>:     sub    $0x10,%rsp
   0x000000000040052e <+8>:     movl   $0x0,-0x8(%rbp)
   0x0000000000400535 <+15>:    movl   $0x0,-0xc(%rbp)
   0x000000000040053c <+22>:    jmp    0x400548 <main+34>
   0x000000000040053e <+24>:    mov    -0xc(%rbp),%eax
=> 0x0000000000400541 <+27>:    add    %eax,-0x8(%rbp)
   0x0000000000400544 <+30>:    addl   $0x1,-0xc(%rbp)
   0x0000000000400548 <+34>:    cmpl   $0x4,-0xc(%rbp)
   0x000000000040054c <+38>:    jle    0x40053e <main+24>
   0x000000000040054e <+40>:    mov    -0x8(%rbp),%eax
   0x0000000000400551 <+43>:    mov    %eax,%edi
   0x0000000000400553 <+45>:    callq  0x400576 <multi2>
   0x0000000000400558 <+50>:    mov    %eax,-0x4(%rbp)
   0x000000000040055b <+53>:    mov    -0x4(%rbp),%eax
   0x000000000040055e <+56>:    mov    %eax,%esi
   0x0000000000400560 <+58>:    mov    $0x400614,%edi
   0x0000000000400565 <+63>:    mov    $0x0,%eax
   0x000000000040056a <+68>:    callq  0x400400 <printf@plt>
   0x000000000040056f <+73>:    mov    $0x0,%eax
   0x0000000000400574 <+78>:    leaveq
   0x0000000000400575 <+79>:    retq
End of assembler dump.
```

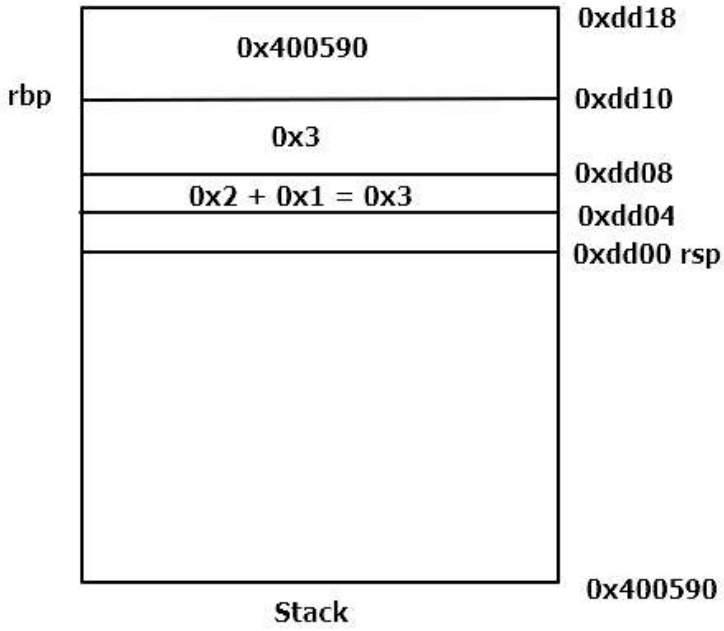


Stack

rsp	0xdd10
rbp	0xdd10
rip	0x400544 (addl $0x1, -0xc(%rbp)
eax	0x2
edi	0x1

Registers in CPU

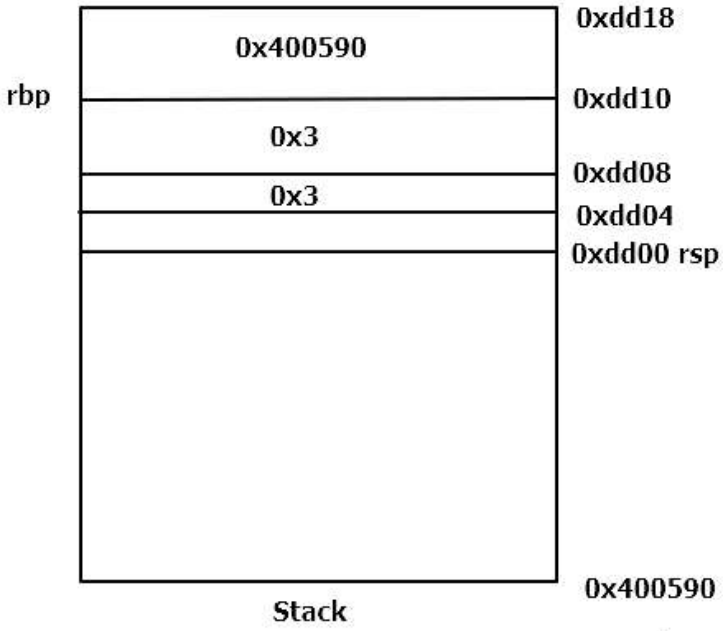
```
Dump of assembler code for function main:
   0x0000000000400526 <+0>:     push   %rbp
   0x0000000000400527 <+1>:     mov    %rsp,%rbp
   0x000000000040052a <+4>:     sub    $0x10,%rsp
   0x000000000040052e <+8>:     movl   $0x0,-0x8(%rbp)
   0x0000000000400535 <+15>:    movl   $0x0,-0xc(%rbp)
   0x000000000040053c <+22>:    jmp    0x400548 <main+34>
   0x000000000040053e <+24>:    mov    -0xc(%rbp),%eax
   0x0000000000400541 <+27>:    add    %eax,-0x8(%rbp)
=> 0x0000000000400544 <+30>:    addl   $0x1,-0xc(%rbp)
   0x0000000000400548 <+34>:    cmpl   $0x4,-0xc(%rbp)
   0x000000000040054c <+38>:    jle    0x40053e <main+24>
   0x000000000040054e <+40>:    mov    -0x8(%rbp),%eax
   0x0000000000400551 <+43>:    mov    %eax,%edi
   0x0000000000400553 <+45>:    callq  0x400576 <multi2>
   0x0000000000400558 <+50>:    mov    %eax,-0x4(%rbp)
   0x000000000040055b <+53>:    mov    -0x4(%rbp),%eax
   0x000000000040055e <+56>:    mov    %eax,%esi
   0x0000000000400560 <+58>:    mov    $0x400614,%edi
   0x0000000000400565 <+63>:    mov    $0x0,%eax
   0x000000000040056a <+68>:    callq  0x400400 <printf@plt>
   0x000000000040056f <+73>:    mov    $0x0,%eax
   0x0000000000400574 <+78>:    leaveq
   0x0000000000400575 <+79>:    retq
End of assembler dump.
```



rsp	0xdd10
rbp	0xdd10
rip	0x400548 (cmpl &0x4, -0xc(rbp))
eax	0x2
edi	0x1

Registers in CPU

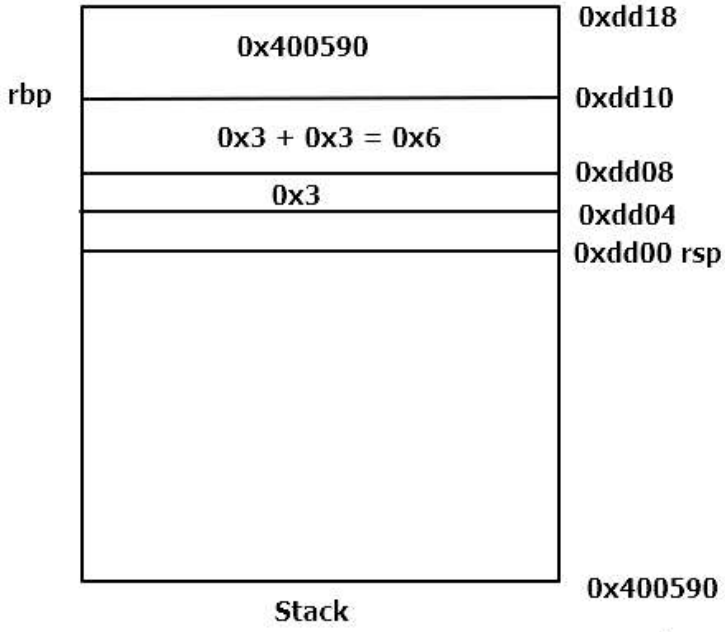
```
Dump of assembler code for function main:
   0x0000000000400526 <+0>:     push   %rbp
   0x0000000000400527 <+1>:     mov    %rsp,%rbp
   0x000000000040052a <+4>:     sub    $0x10,%rsp
   0x000000000040052e <+8>:     movl   $0x0,-0x8(%rbp)
   0x0000000000400535 <+15>:    movl   $0x0,-0xc(%rbp)
   0x000000000040053c <+22>:    jmp    0x400548 <main+34>
   0x000000000040053e <+24>:    mov    -0xc(%rbp),%eax
   0x0000000000400541 <+27>:    add    %eax,-0x8(%rbp)
   0x0000000000400544 <+30>:    addl   $0x1,-0xc(%rbp)
=> 0x0000000000400548 <+34>:    cmpl   $0x4,-0xc(%rbp)
   0x000000000040054c <+38>:    jle    0x40053e <main+24>
   0x000000000040054e <+40>:    mov    -0x8(%rbp),%eax
   0x0000000000400551 <+43>:    mov    %eax,%edi
   0x0000000000400553 <+45>:    callq  0x400576 <multi2>
   0x0000000000400558 <+50>:    mov    %eax,-0x4(%rbp)
   0x000000000040055b <+53>:    mov    -0x4(%rbp),%eax
   0x000000000040055e <+56>:    mov    %eax,%esi
   0x0000000000400560 <+58>:    mov    $0x400614,%edi
   0x0000000000400565 <+63>:    mov    $0x0,%eax
   0x000000000040056a <+68>:    callq  0x400400 <printf@plt>
   0x000000000040056f <+73>:    mov    $0x0,%eax
   0x0000000000400574 <+78>:    leaveq
   0x0000000000400575 <+79>:    retq
End of assembler dump.
```



rsp	0xdd10
rbp	0xdd10
rip	0x400541 (add %eax,-0x8(%rbp))
eax	0x3
edi	0x1

Registers in CPU

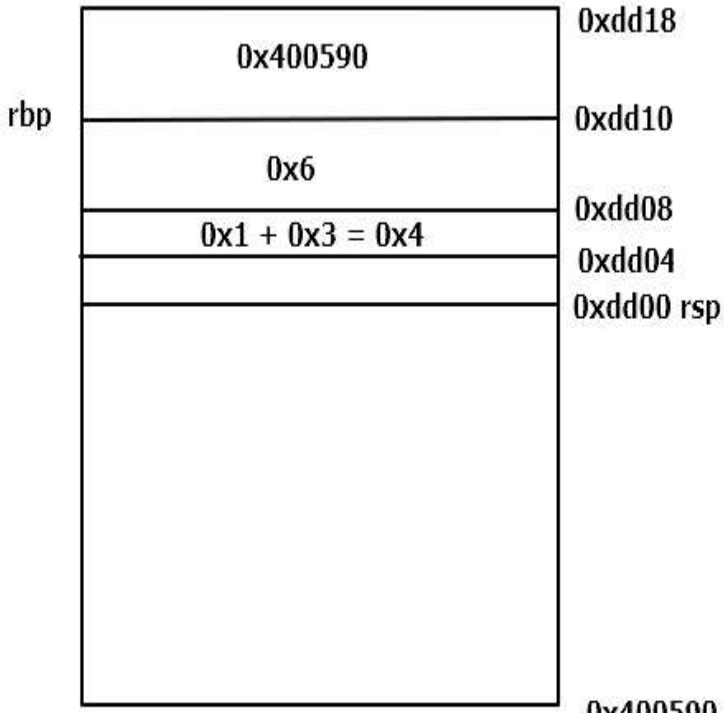
```
Dump of assembler code for function main:
   0x0000000000400526 <+0>:     push   %rbp
   0x0000000000400527 <+1>:     mov    %rsp,%rbp
   0x000000000040052a <+4>:     sub    $0x10,%rsp
   0x000000000040052e <+8>:     movl   $0x0,-0x8(%rbp)
   0x0000000000400535 <+15>:    movl   $0x0,-0xc(%rbp)
   0x000000000040053c <+22>:    jmp    0x400548 <main+34>
   0x000000000040053e <+24>:    mov    -0xc(%rbp),%eax
=> 0x0000000000400541 <+27>:    add    %eax,-0x8(%rbp)
   0x0000000000400544 <+30>:    addl   $0x1,-0xc(%rbp)
   0x0000000000400548 <+34>:    cmpl   $0x4,-0xc(%rbp)
   0x000000000040054c <+38>:    jle    0x40053e <main+24>
   0x000000000040054e <+40>:    mov    -0x8(%rbp),%eax
   0x0000000000400551 <+43>:    mov    %eax,%edi
   0x0000000000400553 <+45>:    callq  0x400576 <multi2>
   0x0000000000400558 <+50>:    mov    %eax,-0x4(%rbp)
   0x000000000040055b <+53>:    mov    -0x4(%rbp),%eax
   0x000000000040055e <+56>:    mov    %eax,%esi
   0x0000000000400560 <+58>:    mov    $0x400614,%edi
   0x0000000000400565 <+63>:    mov    $0x0,%eax
   0x000000000040056a <+68>:    callq  0x400400 <printf@plt>
   0x000000000040056f <+73>:    mov    $0x0,%eax
   0x0000000000400574 <+78>:    leaveq
   0x0000000000400575 <+79>:    retq
End of assembler dump.
```



rsp	0xdd10
rbp	0xdd10
rip	0x400544 (addl $0x1,-0xc(%rbp))
eax	0x3
edi	0x1

Registers in CPU

```
Dump of assembler code for function main:
   0x0000000000400526 <+0>:     push   %rbp
   0x0000000000400527 <+1>:     mov    %rsp,%rbp
   0x000000000040052a <+4>:     sub    $0x10,%rsp
   0x000000000040052e <+8>:     movl   $0x0,-0x8(%rbp)
   0x0000000000400535 <+15>:    movl   $0x0,-0xc(%rbp)
   0x000000000040053c <+22>:    jmp    0x400548 <main+34>
   0x000000000040053e <+24>:    mov    -0xc(%rbp),%eax
   0x0000000000400541 <+27>:    add    %eax,-0x8(%rbp)
=> 0x0000000000400544 <+30>:    addl   $0x1,-0xc(%rbp)
   0x0000000000400548 <+34>:    cmpl   $0x4,-0xc(%rbp)
   0x000000000040054c <+38>:    jle    0x40053e <main+24>
   0x000000000040054e <+40>:    mov    -0x8(%rbp),%eax
   0x0000000000400551 <+43>:    mov    %eax,%edi
   0x0000000000400553 <+45>:    callq  0x400576 <multi2>
   0x0000000000400558 <+50>:    mov    %eax,-0x4(%rbp)
   0x000000000040055b <+53>:    mov    -0x4(%rbp),%eax
   0x000000000040055e <+56>:    mov    %eax,%esi
   0x0000000000400560 <+58>:    mov    $0x400614,%edi
   0x0000000000400565 <+63>:    mov    $0x0,%eax
   0x000000000040056a <+68>:    callq  0x400400 <printf@plt>
   0x000000000040056f <+73>:    mov    $0x0,%eax
   0x0000000000400574 <+78>:    leaveq
   0x0000000000400575 <+79>:    retq
End of assembler dump.
```

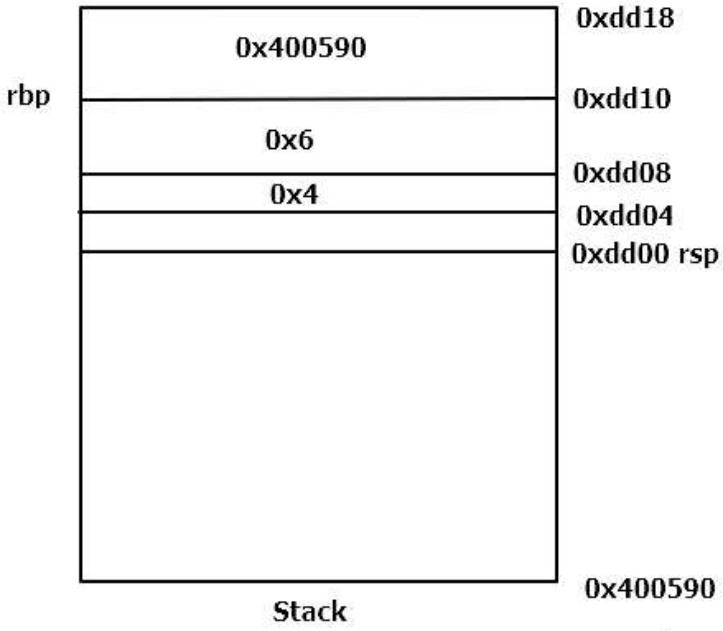


Stack

rsp	0xdd10
rbp	0xdd10
rip	0x400548 (cmpl $0x4,-0xc(%rbp))
eax	0x3
edi	0x1

Registers in CPU

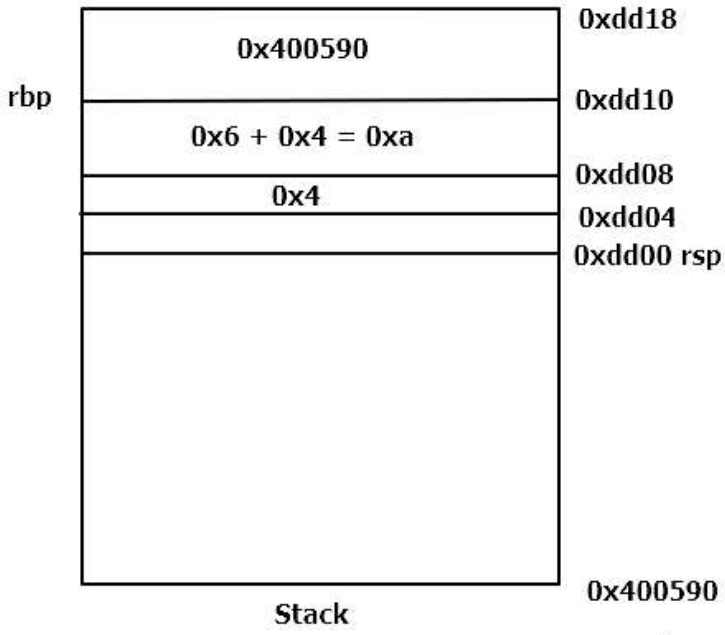
```
Dump of assembler code for function main:
   0x0000000000400526 <+0>:     push   %rbp
   0x0000000000400527 <+1>:     mov    %rsp,%rbp
   0x000000000040052a <+4>:     sub    $0x10,%rsp
   0x000000000040052e <+8>:     movl   $0x0,-0x8(%rbp)
   0x0000000000400535 <+15>:    movl   $0x0,-0xc(%rbp)
   0x000000000040053c <+22>:    jmp    0x400548 <main+34>
   0x000000000040053e <+24>:    mov    -0xc(%rbp),%eax
   0x0000000000400541 <+27>:    add    %eax,-0x8(%rbp)
   0x0000000000400544 <+30>:    addl   $0x1,-0xc(%rbp)
=> 0x0000000000400548 <+34>:    cmpl   $0x4,-0xc(%rbp)
   0x000000000040054c <+38>:    jle    0x40053e <main+24>
   0x000000000040054e <+40>:    mov    -0x8(%rbp),%eax
   0x0000000000400551 <+43>:    mov    %eax,%edi
   0x0000000000400553 <+45>:    callq  0x400576 <multi2>
   0x0000000000400558 <+50>:    mov    %eax,-0x4(%rbp)
   0x000000000040055b <+53>:    mov    -0x4(%rbp),%eax
   0x000000000040055e <+56>:    mov    %eax,%esi
   0x0000000000400560 <+58>:    mov    $0x400614,%edi
   0x0000000000400565 <+63>:    mov    $0x0,%eax
   0x000000000040056a <+68>:    callq  0x400400 <printf@plt>
   0x000000000040056f <+73>:    mov    $0x0,%eax
   0x0000000000400574 <+78>:    leaveq
   0x0000000000400575 <+79>:    retq
End of assembler dump.
```



rsp	0xdd10
rbp	0xdd10
rip	0x400541 (add %eax,-0x8(%rbp))
eax	0x4
edi	0x1

Registers in CPU

```
Dump of assembler code for function main:
   0x0000000000400526 <+0>:     push   %rbp
   0x0000000000400527 <+1>:     mov    %rsp,%rbp
   0x000000000040052a <+4>:     sub    $0x10,%rsp
   0x000000000040052e <+8>:     movl   $0x0,-0x8(%rbp)
   0x0000000000400535 <+15>:    movl   $0x0,-0xc(%rbp)
   0x000000000040053c <+22>:    jmp    0x400548 <main+34>
   0x000000000040053e <+24>:    mov    -0xc(%rbp),%eax
=> 0x0000000000400541 <+27>:    add    %eax,-0x8(%rbp)
   0x0000000000400544 <+30>:    addl   $0x1,-0xc(%rbp)
   0x0000000000400548 <+34>:    cmpl   $0x4,-0xc(%rbp)
   0x000000000040054c <+38>:    jle    0x40053e <main+24>
   0x000000000040054e <+40>:    mov    -0x8(%rbp),%eax
   0x0000000000400551 <+43>:    mov    %eax,%edi
   0x0000000000400553 <+45>:    callq  0x400576 <multi2>
   0x0000000000400558 <+50>:    mov    %eax,-0x4(%rbp)
   0x000000000040055b <+53>:    mov    -0x4(%rbp),%eax
   0x000000000040055e <+56>:    mov    %eax,%esi
   0x0000000000400560 <+58>:    mov    $0x400614,%edi
   0x0000000000400565 <+63>:    mov    $0x0,%eax
   0x000000000040056a <+68>:    callq  0x400400 <printf@plt>
   0x000000000040056f <+73>:    mov    $0x0,%eax
   0x0000000000400574 <+78>:    leaveq
   0x0000000000400575 <+79>:    retq
End of assembler dump.
```

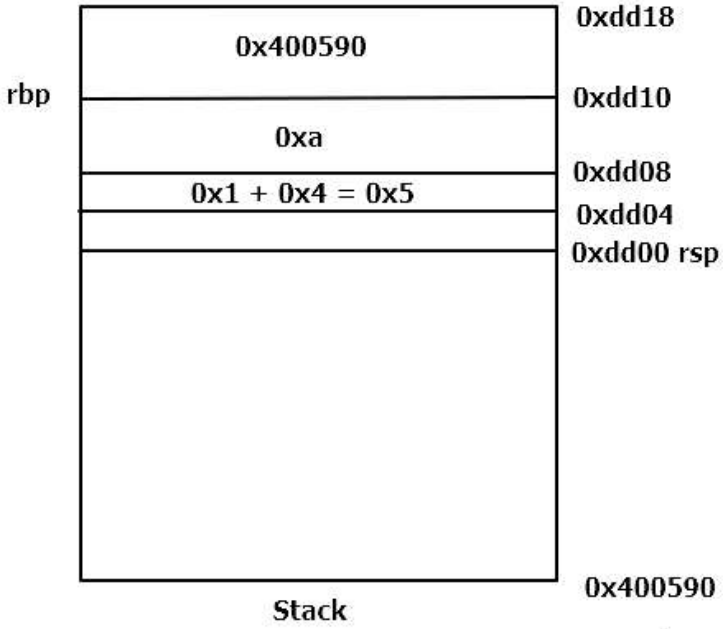


Stack

rsp	0xdd10
rbp	0xdd10
rip	0x400544 (addl $0x1,-0xc(%rbp)
eax	0x4
edi	0x1

Registers in CPU

```
Dump of assembler code for function main:
   0x0000000000400526 <+0>:     push   %rbp
   0x0000000000400527 <+1>:     mov    %rsp,%rbp
   0x000000000040052a <+4>:     sub    $0x10,%rsp
   0x000000000040052e <+8>:     movl   $0x0,-0x8(%rbp)
   0x0000000000400535 <+15>:    movl   $0x0,-0xc(%rbp)
   0x000000000040053c <+22>:    jmp    0x400548 <main+34>
   0x000000000040053e <+24>:    mov    -0xc(%rbp),%eax
   0x0000000000400541 <+27>:    add    %eax,-0x8(%rbp)
=> 0x0000000000400544 <+30>:    addl   $0x1,-0xc(%rbp)
   0x0000000000400548 <+34>:    cmpl   $0x4,-0xc(%rbp)
   0x000000000040054c <+38>:    jle    0x40053e <main+24>
   0x000000000040054e <+40>:    mov    -0x8(%rbp),%eax
   0x0000000000400551 <+43>:    mov    %eax,%edi
   0x0000000000400553 <+45>:    callq  0x400576 <multi2>
   0x0000000000400558 <+50>:    mov    %eax,-0x4(%rbp)
   0x000000000040055b <+53>:    mov    -0x4(%rbp),%eax
   0x000000000040055e <+56>:    mov    %eax,%esi
   0x0000000000400560 <+58>:    mov    $0x400614,%edi
   0x0000000000400565 <+63>:    mov    $0x0,%eax
   0x000000000040056a <+68>:    callq  0x400400 <printf@plt>
   0x000000000040056f <+73>:    mov    $0x0,%eax
   0x0000000000400574 <+78>:    leaveq
   0x0000000000400575 <+79>:    retq
End of assembler dump.
```

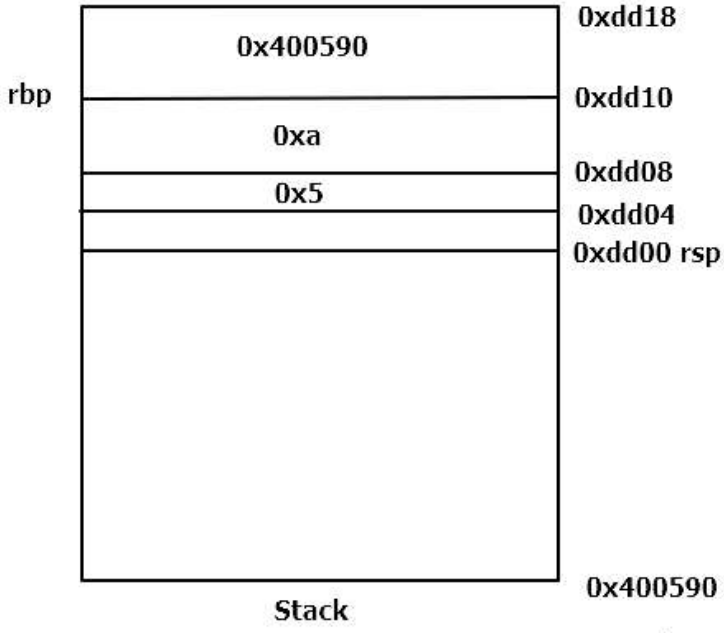


Stack

rsp	0xdd10
rbp	0xdd10
rip	0x400548 (cmpl $0x4,-0xc(%rbp)
eax	0x4
edi	0x1

Registers in CPU

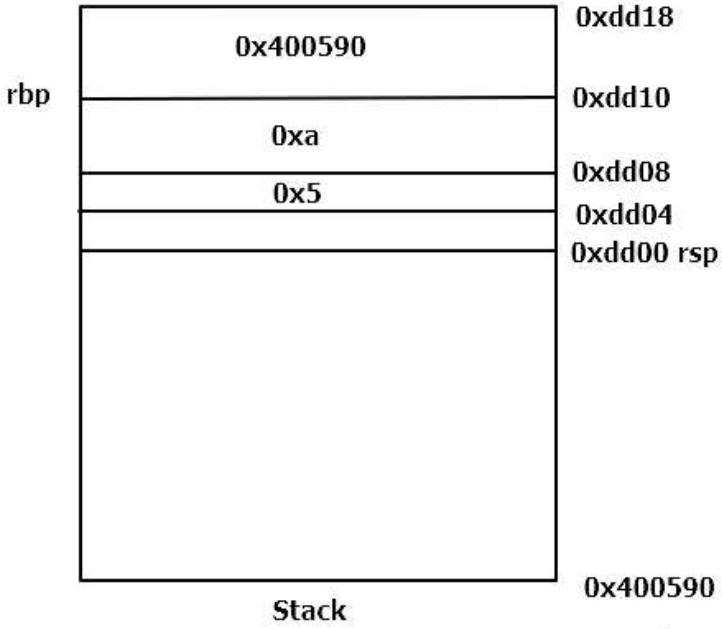
```
Dump of assembler code for function main:
   0x0000000000400526 <+0>:     push   %rbp
   0x0000000000400527 <+1>:     mov    %rsp,%rbp
   0x000000000040052a <+4>:     sub    $0x10,%rsp
   0x000000000040052e <+8>:     movl   $0x0,-0x8(%rbp)
   0x0000000000400535 <+15>:    movl   $0x0,-0xc(%rbp)
   0x000000000040053c <+22>:    jmp    0x400548 <main+34>
   0x000000000040053e <+24>:    mov    -0xc(%rbp),%eax
   0x0000000000400541 <+27>:    add    %eax,-0x8(%rbp)
   0x0000000000400544 <+30>:    addl   $0x1,-0xc(%rbp)
=> 0x0000000000400548 <+34>:    cmpl   $0x4,-0xc(%rbp)
   0x000000000040054c <+38>:    jle    0x40053e <main+24>
   0x000000000040054e <+40>:    mov    -0x8(%rbp),%eax
   0x0000000000400551 <+43>:    mov    %eax,%edi
   0x0000000000400553 <+45>:    callq  0x400576 <multi2>
   0x0000000000400558 <+50>:    mov    %eax,-0x4(%rbp)
   0x000000000040055b <+53>:    mov    -0x4(%rbp),%eax
   0x000000000040055e <+56>:    mov    %eax,%esi
   0x0000000000400560 <+58>:    mov    $0x400614,%edi
   0x0000000000400565 <+63>:    mov    $0x0,%eax
   0x000000000040056a <+68>:    callq  0x400400 <printf@plt>
   0x000000000040056f <+73>:    mov    $0x0,%eax
   0x0000000000400574 <+78>:    leaveq
   0x0000000000400575 <+79>:    retq
End of assembler dump.
```



rsp	0xdd10
rbp	0xdd10
rip	0x40054c (jle 0x40053 <main+24>)
eax	0x4
edi	0x1

Registers in CPU

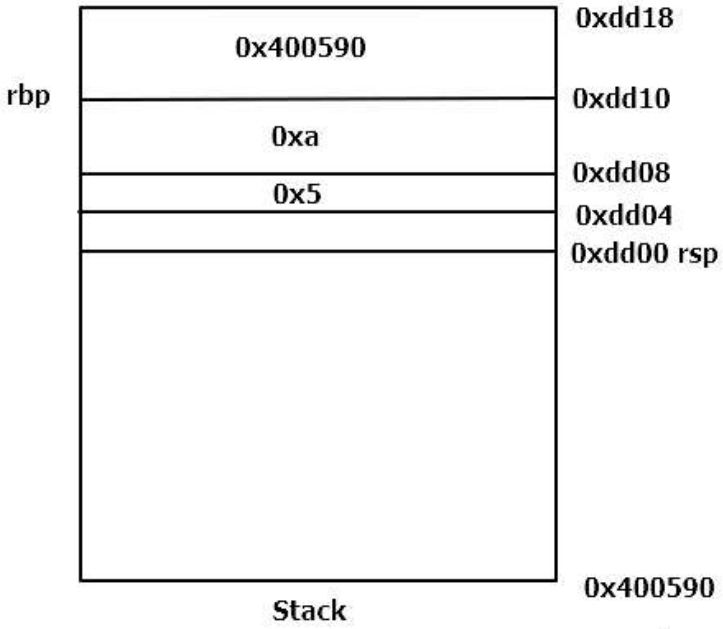
```
Dump of assembler code for function main:
   0x0000000000400526 <+0>:     push   %rbp
   0x0000000000400527 <+1>:     mov    %rsp,%rbp
   0x000000000040052a <+4>:     sub    $0x10,%rsp
   0x000000000040052e <+8>:     movl   $0x0,-0x8(%rbp)
   0x0000000000400535 <+15>:    movl   $0x0,-0xc(%rbp)
   0x000000000040053c <+22>:    jmp    0x400548 <main+34>
   0x000000000040053e <+24>:    mov    -0xc(%rbp),%eax
   0x0000000000400541 <+27>:    add    %eax,-0x8(%rbp)
   0x0000000000400544 <+30>:    addl   $0x1,-0xc(%rbp)
   0x0000000000400548 <+34>:    cmpl   $0x4,-0xc(%rbp)
=> 0x000000000040054c <+38>:    jle    0x40053e <main+24>
   0x000000000040054e <+40>:    mov    -0x8(%rbp),%eax
   0x0000000000400551 <+43>:    mov    %eax,%edi
   0x0000000000400553 <+45>:    callq  0x400576 <multi2>
   0x0000000000400558 <+50>:    mov    %eax,-0x4(%rbp)
   0x000000000040055b <+53>:    mov    -0x4(%rbp),%eax
   0x000000000040055e <+56>:    mov    %eax,%esi
   0x0000000000400560 <+58>:    mov    $0x400614,%edi
   0x0000000000400565 <+63>:    mov    $0x0,%eax
   0x000000000040056a <+68>:    callq  0x400400 <printf@plt>
   0x000000000040056f <+73>:    mov    $0x0,%eax
   0x0000000000400574 <+78>:    leaveq
   0x0000000000400575 <+79>:    retq
End of assembler dump.
```



rsp	0xdd10
rbp	0xdd10
rip	0x40054e (mov -0x8(%rbp),%ea:
eax	0x4
edi	0x1

Registers in CPU

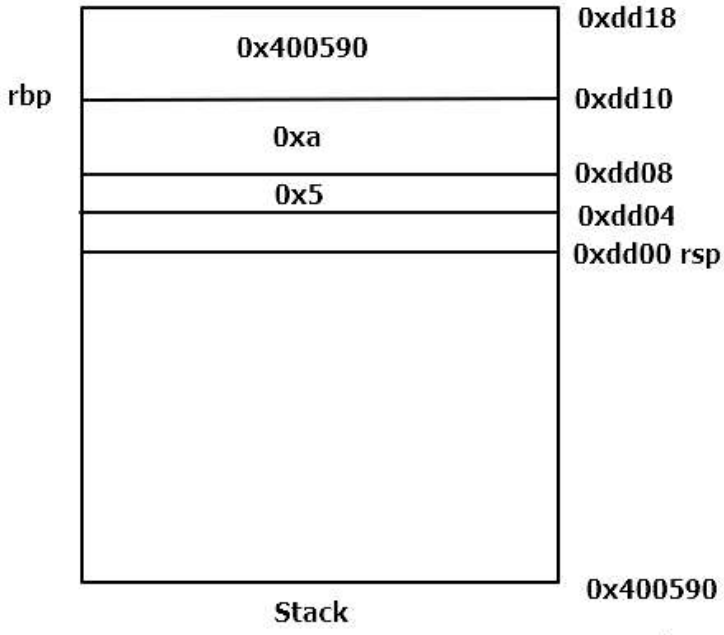
```
Dump of assembler code for function main:
   0x0000000000400526 <+0>:     push   %rbp
   0x0000000000400527 <+1>:     mov    %rsp,%rbp
   0x000000000040052a <+4>:     sub    $0x10,%rsp
   0x000000000040052e <+8>:     movl   $0x0,-0x8(%rbp)
   0x0000000000400535 <+15>:    movl   $0x0,-0xc(%rbp)
   0x000000000040053c <+22>:    jmp    0x400548 <main+34>
   0x000000000040053e <+24>:    mov    -0xc(%rbp),%eax
   0x0000000000400541 <+27>:    add    %eax,-0x8(%rbp)
   0x0000000000400544 <+30>:    addl   $0x1,-0xc(%rbp)
   0x0000000000400548 <+34>:    cmpl   $0x4,-0xc(%rbp)
   0x000000000040054c <+38>:    jle    0x40053e <main+24>
=> 0x000000000040054e <+40>:    mov    -0x8(%rbp),%eax
   0x0000000000400551 <+43>:    mov    %eax,%edi
   0x0000000000400553 <+45>:    callq  0x400576 <multi2>
   0x0000000000400558 <+50>:    mov    %eax,-0x4(%rbp)
   0x000000000040055b <+53>:    mov    -0x4(%rbp),%eax
   0x000000000040055e <+56>:    mov    %eax,%esi
   0x0000000000400560 <+58>:    mov    $0x400614,%edi
   0x0000000000400565 <+63>:    mov    $0x0,%eax
   0x000000000040056a <+68>:    callq  0x400400 <printf@plt>
   0x000000000040056f <+73>:    mov    $0x0,%eax
   0x0000000000400574 <+78>:    leaveq
   0x0000000000400575 <+79>:    retq
End of assembler dump.
```



rsp	0xdd10
rbp	0xdd10
rip	0x400551 (mov %eax,%edi)
eax	0xa
edi	0x1

Registers in CPU

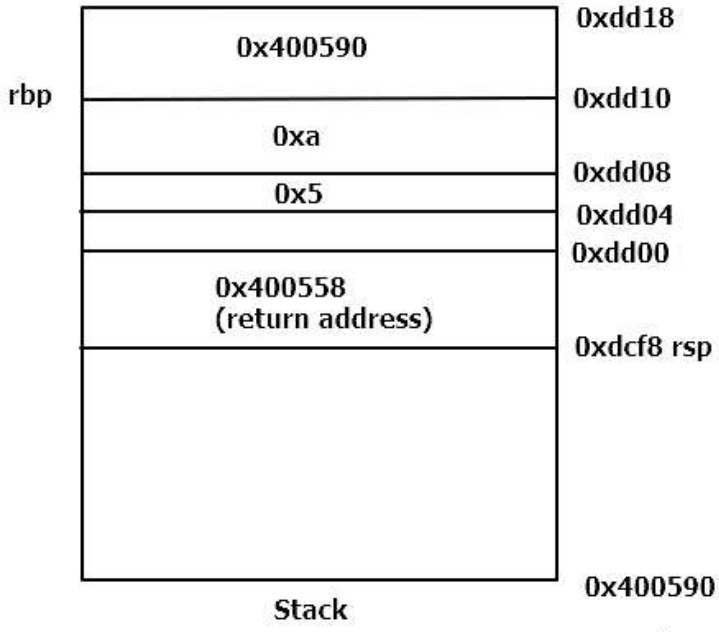
```
Dump of assembler code for function main:
   0x0000000000400526 <+0>:     push   %rbp
   0x0000000000400527 <+1>:     mov    %rsp,%rbp
   0x000000000040052a <+4>:     sub    $0x10,%rsp
   0x000000000040052e <+8>:     movl   $0x0,-0x8(%rbp)
   0x0000000000400535 <+15>:    movl   $0x0,-0xc(%rbp)
   0x000000000040053c <+22>:    jmp    0x400548 <main+34>
   0x000000000040053e <+24>:    mov    -0xc(%rbp),%eax
   0x0000000000400541 <+27>:    add    %eax,-0x8(%rbp)
   0x0000000000400544 <+30>:    addl   $0x1,-0xc(%rbp)
   0x0000000000400548 <+34>:    cmpl   $0x4,-0xc(%rbp)
   0x000000000040054c <+38>:    jle    0x40053e <main+24>
   0x000000000040054e <+40>:    mov    -0x8(%rbp),%eax
=> 0x0000000000400551 <+43>:    mov    %eax,%edi
   0x0000000000400553 <+45>:    callq  0x400576 <multi2>
   0x0000000000400558 <+50>:    mov    %eax,-0x4(%rbp)
   0x000000000040055b <+53>:    mov    -0x4(%rbp),%eax
   0x000000000040055e <+56>:    mov    %eax,%esi
   0x0000000000400560 <+58>:    mov    $0x400614,%edi
   0x0000000000400565 <+63>:    mov    $0x0,%eax
   0x000000000040056a <+68>:    callq  0x400400 <printf@plt>
   0x000000000040056f <+73>:    mov    $0x0,%eax
   0x0000000000400574 <+78>:    leaveq
   0x0000000000400575 <+79>:    retq
End of assembler dump.
```



rsp	0xdd10
rbp	0xdd10
rip	0x400553 (callq 0x400576 <multi2>)
eax	0xa
edi	0xa

Registers in CPU

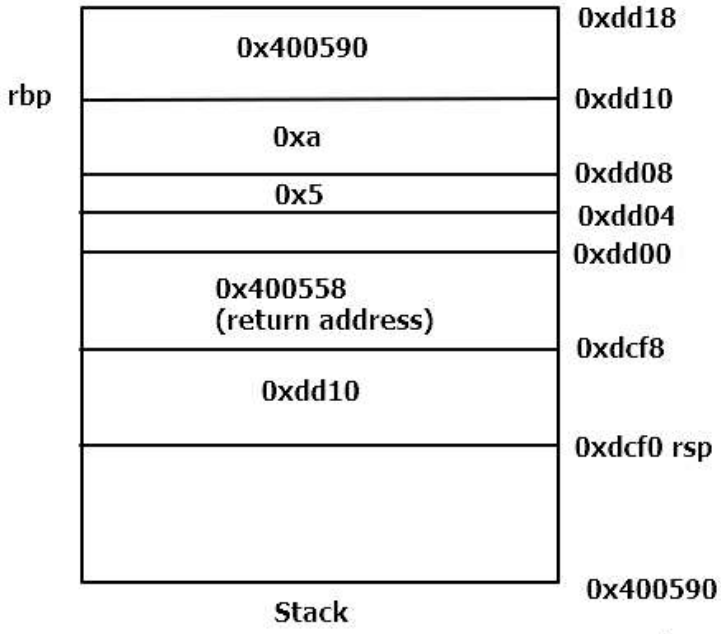
```
Dump of assembler code for function main:
   0x0000000000400526 <+0>:     push   %rbp
   0x0000000000400527 <+1>:     mov    %rsp,%rbp
   0x000000000040052a <+4>:     sub    $0x10,%rsp
   0x000000000040052e <+8>:     movl   $0x0,-0x8(%rbp)
   0x0000000000400535 <+15>:    movl   $0x0,-0xc(%rbp)
   0x000000000040053c <+22>:    jmp    0x400548 <main+34>
   0x000000000040053e <+24>:    mov    -0xc(%rbp),%eax
   0x0000000000400541 <+27>:    add    %eax,-0x8(%rbp)
   0x0000000000400544 <+30>:    addl   $0x1,-0xc(%rbp)
   0x0000000000400548 <+34>:    cmpl   $0x4,-0xc(%rbp)
   0x000000000040054c <+38>:    jle    0x40053e <main+24>
   0x000000000040054e <+40>:    mov    -0x8(%rbp),%eax
   0x0000000000400551 <+43>:    mov    %eax,%edi
=> 0x0000000000400553 <+45>:    callq  0x400576 <multi2>
   0x0000000000400558 <+50>:    mov    %eax,-0x4(%rbp)
   0x000000000040055b <+53>:    mov    -0x4(%rbp),%eax
   0x000000000040055e <+56>:    mov    %eax,%esi
   0x0000000000400560 <+58>:    mov    $0x400614,%edi
   0x0000000000400565 <+63>:    mov    $0x0,%eax
   0x000000000040056a <+68>:    callq  0x400400 <printf@plt>
   0x000000000040056f <+73>:    mov    $0x0,%eax
   0x0000000000400574 <+78>:    leaveq
   0x0000000000400575 <+79>:    retq
End of assembler dump.
```



rsp	0xdcf8
rbp	0xdd10
rip	0x400576 (push %rbp)
eax	0xa
edi	0xa

Registers in CPU

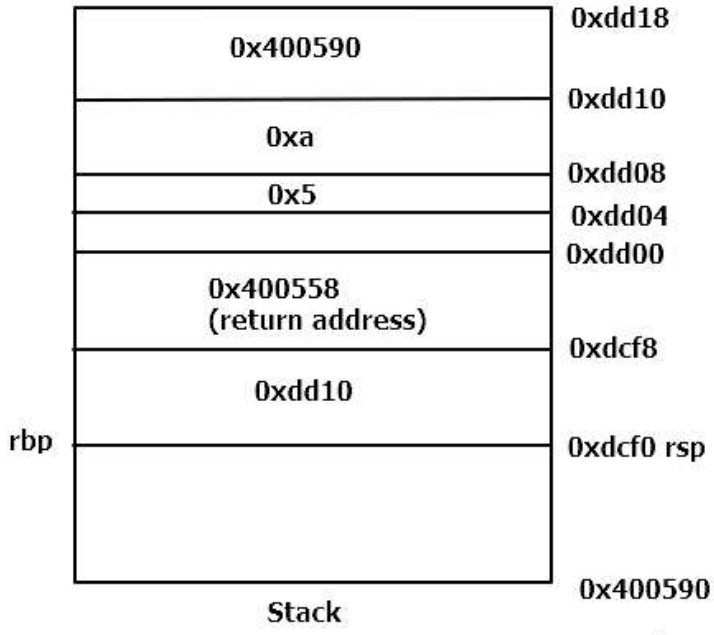
```
Dump of assembler code for function multi2:
=> 0x0000000000400576 <+0>:     push   %rbp
   0x0000000000400577 <+1>:     mov    %rsp,%rbp
   0x000000000040057a <+4>:     mov    %edi,-0x4(%rbp)
   0x000000000040057d <+7>:     mov    -0x4(%rbp),%eax
   0x0000000000400580 <+10>:    add    %eax,%eax
   0x0000000000400582 <+12>:    pop    %rbp
   0x0000000000400583 <+13>:    retq
End of assembler dump.
```



rsp	0xdcf0
rbp	0xdd10
rip	0x400577 (mov %rsp,%rbp)
eax	0xa
edi	0xa

Registers in CPU

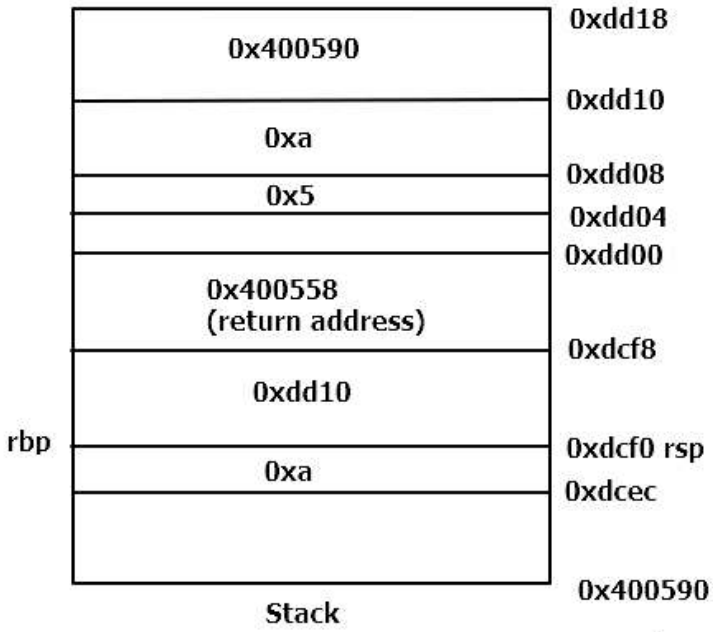
```
Dump of assembler code for function multi2:
   0x0000000000400576 <+0>:     push   %rbp
=> 0x0000000000400577 <+1>:     mov    %rsp,%rbp
   0x000000000040057a <+4>:     mov    %edi,-0x4(%rbp)
   0x000000000040057d <+7>:     mov    -0x4(%rbp),%eax
   0x0000000000400580 <+10>:    add    %eax,%eax
   0x0000000000400582 <+12>:    pop    %rbp
   0x0000000000400583 <+13>:    retq
End of assembler dump.
```



rsp	0xdcf0
rbp	0xdd10
rip	0x40057a (mov %edi,-0x4(%rbp))
eax	0xa
edi	0xa

Registers in CPU

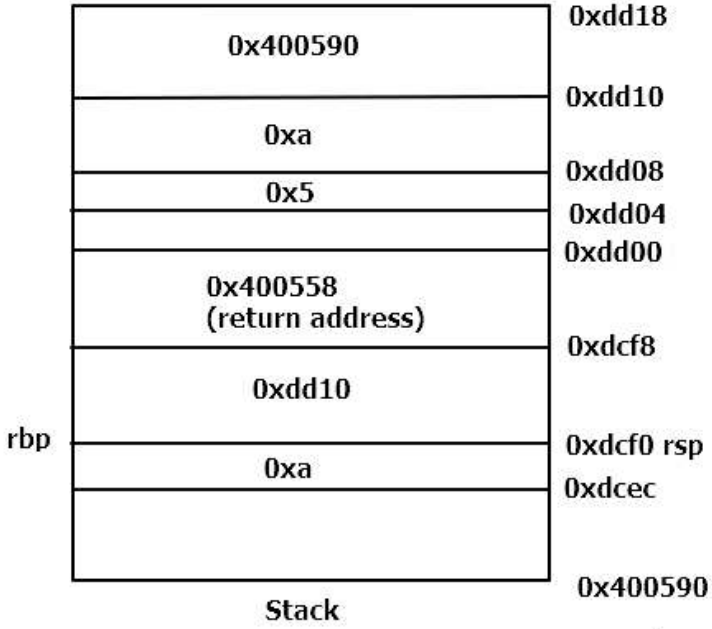
```
Dump of assembler code for function multi2:
   0x0000000000400576 <+0>:     push   %rbp
   0x0000000000400577 <+1>:     mov    %rsp,%rbp
=> 0x000000000040057a <+4>:     mov    %edi,-0x4(%rbp)
   0x000000000040057d <+7>:     mov    -0x4(%rbp),%eax
   0x0000000000400580 <+10>:    add    %eax,%eax
   0x0000000000400582 <+12>:    pop    %rbp
   0x0000000000400583 <+13>:    retq
End of assembler dump.
```



rsp	0xdcf0
rbp	0xdd10
rip	0x40057d (mov -0x4(%rbp),%eax)
eax	0xa
edi	0xa

Registers in CPU

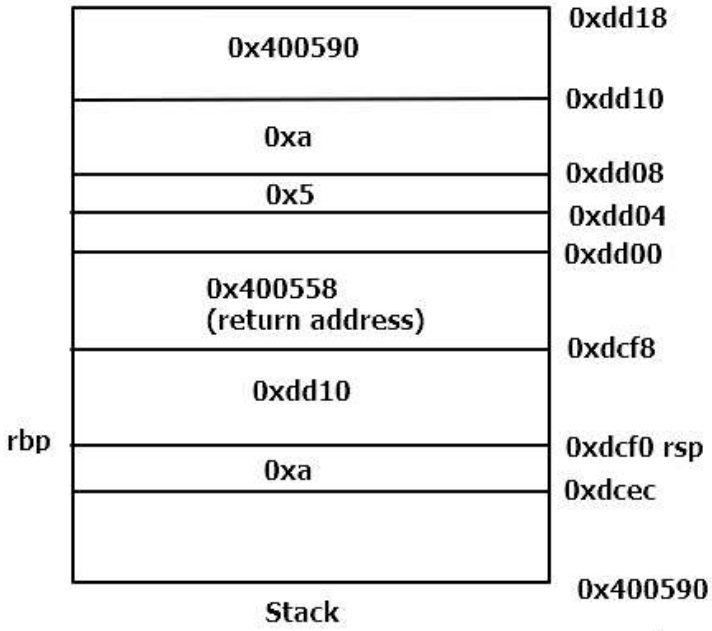
```
Dump of assembler code for function multi2:
   0x0000000000400576 <+0>:     push   %rbp
   0x0000000000400577 <+1>:     mov    %rsp,%rbp
   0x000000000040057a <+4>:     mov    %edi,-0x4(%rbp)
=> 0x000000000040057d <+7>:     mov    -0x4(%rbp),%eax
   0x0000000000400580 <+10>:    add    %eax,%eax
   0x0000000000400582 <+12>:    pop    %rbp
   0x0000000000400583 <+13>:    retq
End of assembler dump.
```



rsp	0xdcf0
rbp	0xdd10
rip	0x400580 (add %eax,%eax)
eax	0xa
edi	0xa

Registers in CPU

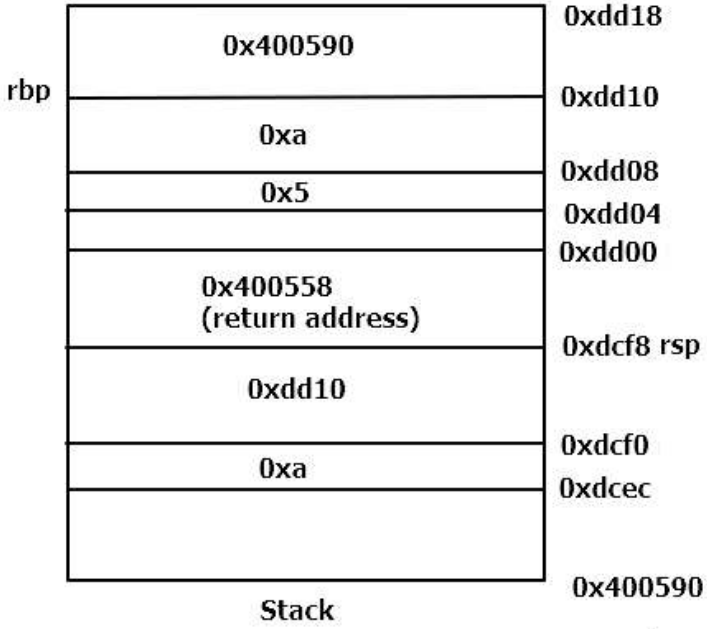
```
Dump of assembler code for function multi2:
   0x0000000000400576 <+0>:     push   %rbp
   0x0000000000400577 <+1>:     mov    %rsp,%rbp
   0x000000000040057a <+4>:     mov    %edi,-0x4(%rbp)
   0x000000000040057d <+7>:     mov    -0x4(%rbp),%eax
=> 0x0000000000400580 <+10>:    add    %eax,%eax
   0x0000000000400582 <+12>:    pop    %rbp
   0x0000000000400583 <+13>:    retq
End of assembler dump.
```



rsp	0xdcf0
rbp	0xdd10
rip	0x400582 (pop %rbp)
eax	0xa + 0xa = 0x14
edi	0xa

Registers in CPU

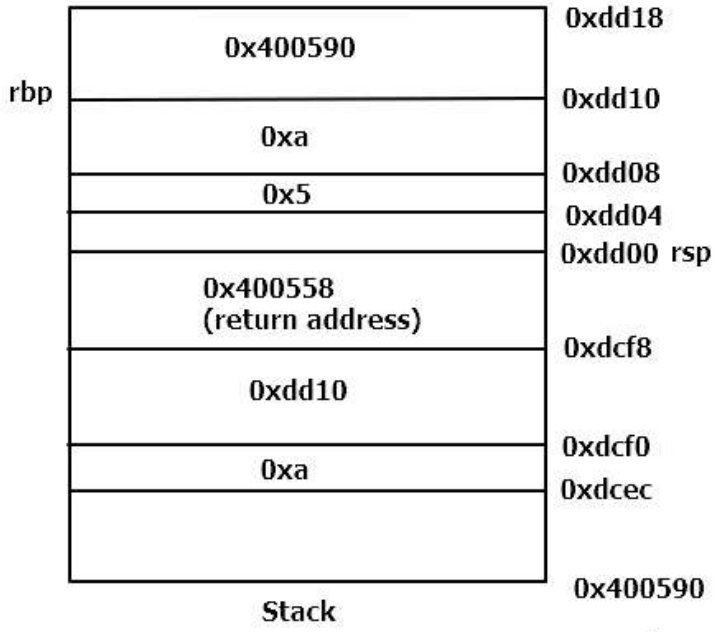
```
Dump of assembler code for function multi2:
   0x0000000000400576 <+0>:     push   %rbp
   0x0000000000400577 <+1>:     mov    %rsp,%rbp
   0x000000000040057a <+4>:     mov    %edi,-0x4(%rbp)
   0x000000000040057d <+7>:     mov    -0x4(%rbp),%eax
   0x0000000000400580 <+10>:    add    %eax,%eax
=> 0x0000000000400582 <+12>:    pop    %rbp
   0x0000000000400583 <+13>:    retq
End of assembler dump.
```



rsp	0xdcf8
rbp	0xdd10
rip	0x400583 (retq)
eax	0x14
edi	0xa

Registers in CPU

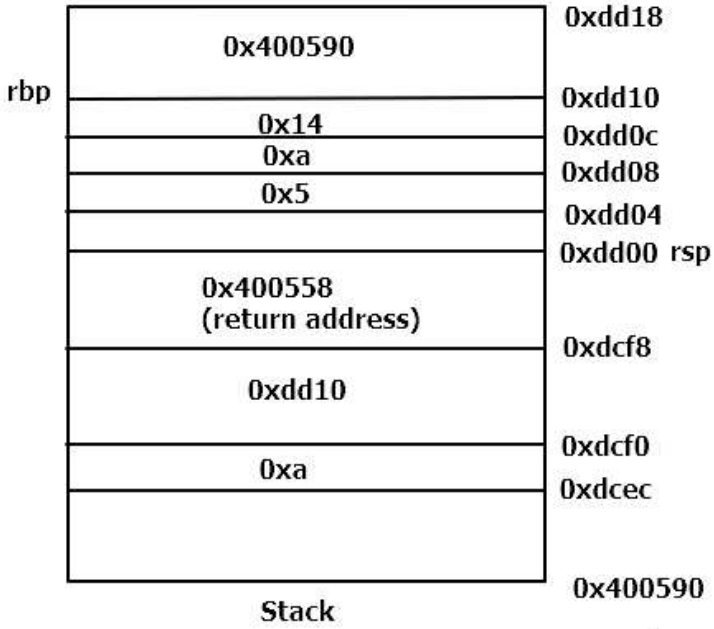
```
Dump of assembler code for function multi2:
   0x0000000000400576 <+0>:     push   %rbp
   0x0000000000400577 <+1>:     mov    %rsp,%rbp
   0x000000000040057a <+4>:     mov    %edi,-0x4(%rbp)
   0x000000000040057d <+7>:     mov    -0x4(%rbp),%eax
   0x0000000000400580 <+10>:    add    %eax,%eax
   0x0000000000400582 <+12>:    pop    %rbp
=> 0x0000000000400583 <+13>:    retq
End of assembler dump.
```



rsp	0xdcf8
rbp	0xdd00
rip	0x400558(return addresss) (mov -0x8(%rbp),-0x4(%rbp)
eax	0x14
edi	0xa

Registers in CPU

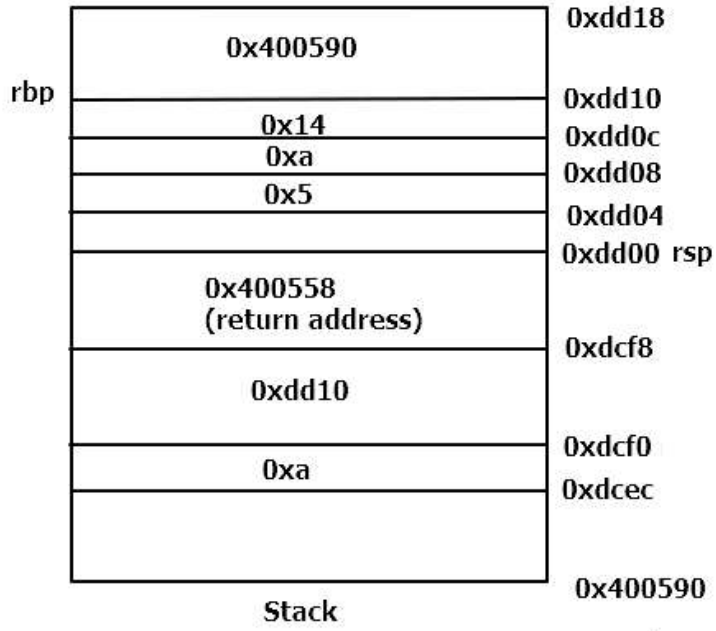
```
Dump of assembler code for function main:
   0x0000000000400526 <+0>:     push   %rbp
   0x0000000000400527 <+1>:     mov    %rsp,%rbp
   0x000000000040052a <+4>:     sub    $0x10,%rsp
   0x000000000040052e <+8>:     movl   $0x0,-0x8(%rbp)
   0x0000000000400535 <+15>:    movl   $0x0,-0xc(%rbp)
   0x000000000040053c <+22>:    jmp    0x400548 <main+34>
   0x000000000040053e <+24>:    mov    -0xc(%rbp),%eax
   0x0000000000400541 <+27>:    add    %eax,-0x8(%rbp)
   0x0000000000400544 <+30>:    addl   $0x1,-0xc(%rbp)
   0x0000000000400548 <+34>:    cmpl   $0x4,-0xc(%rbp)
   0x000000000040054c <+38>:    jle    0x40053e <main+24>
   0x000000000040054e <+40>:    mov    -0x8(%rbp),%eax
   0x0000000000400551 <+43>:    mov    %eax,%edi
   0x0000000000400553 <+45>:    callq  0x400576 <multi2>
=> 0x0000000000400558 <+50>:    mov    %eax,-0x4(%rbp)
   0x000000000040055b <+53>:    mov    -0x4(%rbp),%eax
   0x000000000040055e <+56>:    mov    %eax,%esi
   0x0000000000400560 <+58>:    mov    $0x400614,%edi
   0x0000000000400565 <+63>:    mov    $0x0,%eax
   0x000000000040056a <+68>:    callq  0x400400 <printf@plt>
   0x000000000040056f <+73>:    mov    $0x0,%eax
   0x0000000000400574 <+78>:    leaveq
   0x0000000000400575 <+79>:    retq
End of assembler dump.
```



rsp	0xdcf8
rbp	0xdd00
rip	0x40055b (mov -0x4(%rbp),%eax)
eax	0x14
edi	0xa

Registers in CPU

```
Dump of assembler code for function main:
   0x0000000000400526 <+0>:     push   %rbp
   0x0000000000400527 <+1>:     mov    %rsp,%rbp
   0x000000000040052a <+4>:     sub    $0x10,%rsp
   0x000000000040052e <+8>:     movl   $0x0,-0x8(%rbp)
   0x0000000000400535 <+15>:    movl   $0x0,-0xc(%rbp)
   0x000000000040053c <+22>:    jmp    0x400548 <main+34>
   0x000000000040053e <+24>:    mov    -0xc(%rbp),%eax
   0x0000000000400541 <+27>:    add    %eax,-0x8(%rbp)
   0x0000000000400544 <+30>:    addl   $0x1,-0xc(%rbp)
   0x0000000000400548 <+34>:    cmpl   $0x4,-0xc(%rbp)
   0x000000000040054c <+38>:    jle    0x40053e <main+24>
   0x000000000040054e <+40>:    mov    -0x8(%rbp),%eax
   0x0000000000400551 <+43>:    mov    %eax,%edi
   0x0000000000400553 <+45>:    callq  0x400576 <multi2>
   0x0000000000400558 <+50>:    mov    %eax,-0x4(%rbp)
=> 0x000000000040055b <+53>:    mov    -0x4(%rbp),%eax
   0x000000000040055e <+56>:    mov    %eax,%esi
   0x0000000000400560 <+58>:    mov    $0x400614,%edi
   0x0000000000400565 <+63>:    mov    $0x0,%eax
   0x000000000040056a <+68>:    callq  0x400400 <printf@plt>
   0x000000000040056f <+73>:    mov    $0x0,%eax
   0x0000000000400574 <+78>:    leaveq
   0x0000000000400575 <+79>:    retq
End of assembler dump.
```



Stack

rsp	0xdcf8
rbp	0xdd00
rip	0x40055e (move %eax,%esi)
eax	0x14
edi	0xa

Registers in CPU

```
Dump of assembler code for function main:
   0x0000000000400526 <+0>:     push   %rbp
   0x0000000000400527 <+1>:     mov    %rsp,%rbp
   0x000000000040052a <+4>:     sub    $0x10,%rsp
   0x000000000040052e <+8>:     movl   $0x0,-0x8(%rbp)
   0x0000000000400535 <+15>:    movl   $0x0,-0xc(%rbp)
   0x000000000040053c <+22>:    jmp    0x400548 <main+34>
   0x000000000040053e <+24>:    mov    -0xc(%rbp),%eax
   0x0000000000400541 <+27>:    add    %eax,-0x8(%rbp)
   0x0000000000400544 <+30>:    addl   $0x1,-0xc(%rbp)
   0x0000000000400548 <+34>:    cmpl   $0x4,-0xc(%rbp)
   0x000000000040054c <+38>:    jle    0x40053e <main+24>
   0x000000000040054e <+40>:    mov    -0x8(%rbp),%eax
   0x0000000000400551 <+43>:    mov    %eax,%edi
   0x0000000000400553 <+45>:    callq  0x400576 <multi2>
   0x0000000000400558 <+50>:    mov    %eax,-0x4(%rbp)
   0x000000000040055b <+53>:    mov    -0x4(%rbp),%eax
=> 0x000000000040055e <+56>:    mov    %eax,%esi
   0x0000000000400560 <+58>:    mov    $0x400614,%edi
   0x0000000000400565 <+63>:    mov    $0x0,%eax
   0x000000000040056a <+68>:    callq  0x400400 <printf@plt>
   0x000000000040056f <+73>:    mov    $0x0,%eax
   0x0000000000400574 <+78>:    leaveq
   0x0000000000400575 <+79>:    retq
End of assembler dump.
```