香港中文大學
The Chinese University of Hong Kong

CSCI2510 Computer Organization

Tutorial 07: Subroutine in MASM

Bentian Jiang

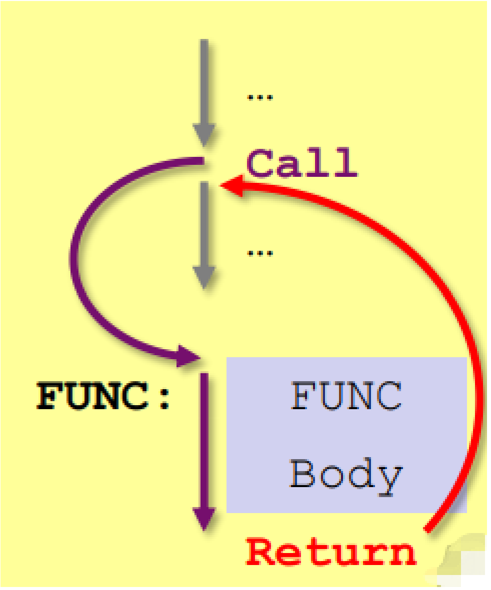
btjiang@cse.cuhk.edu.hk

Outline

- Subroutine Revisited
- Stack Frame
- Greatest Common Divisor

Subroutine Review

- Basic concepts:
  - When a program branches to a subroutine we say that it is **calling** the subroutine.
  - After a subroutine calling, the subroutine is said to **return** to the program that called it.
    - Continuing immediately after the instruction that called the subroutine.
  - Provision must be made for returning to the appropriate location.
    - the contents of the PC must be saved by the call instruction to enable correct return to the calling program



Why Subroutine?

- Divide/Decrease and conquer
- Reuse codes
- Make variable namespace clean

Divide/Decrease and conquer

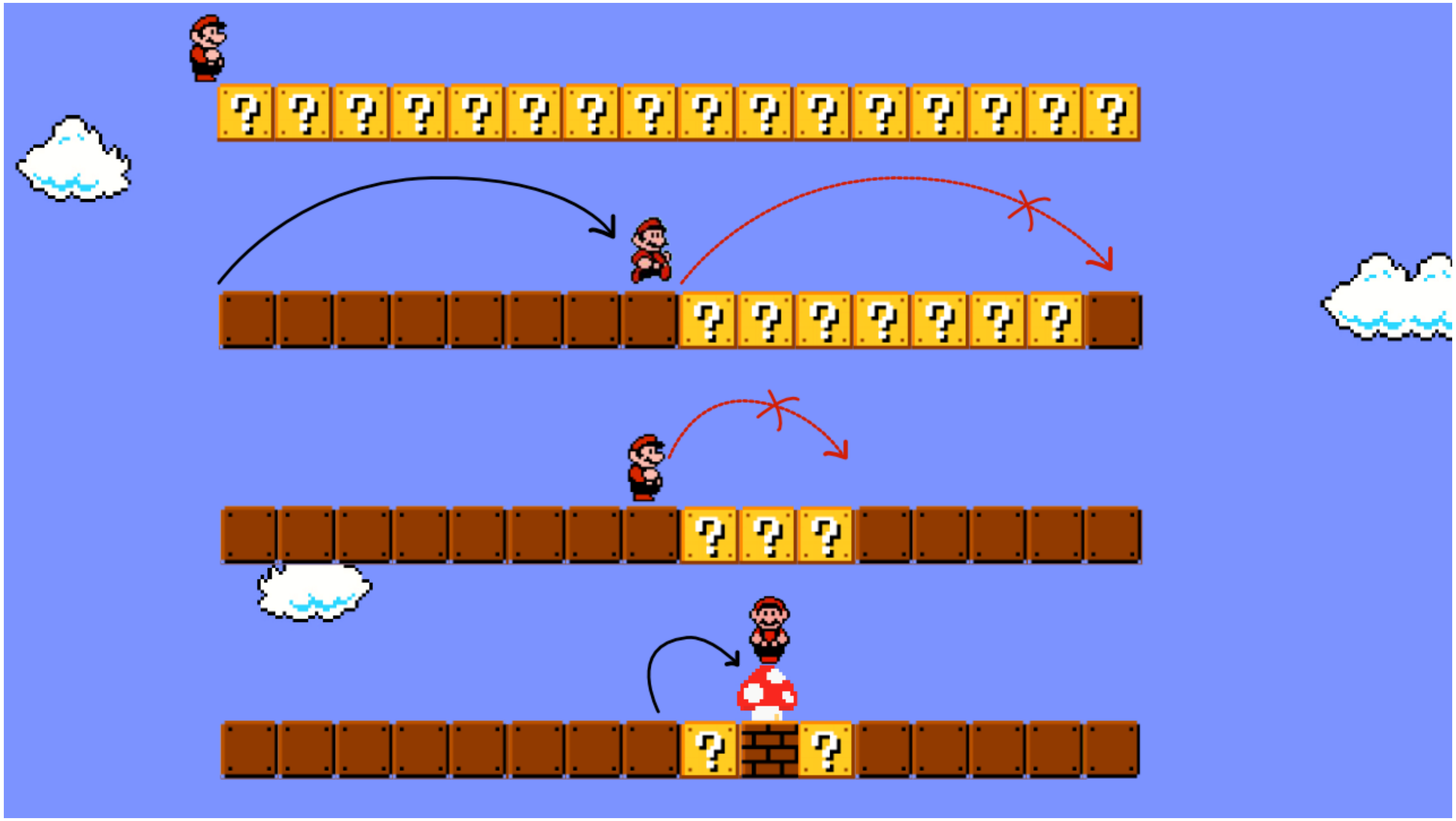
- Philosophy of multi-branched recursion

- A divide/decrease and conquer algorithm works by
  - Divide: recursively breaking down a problem into two or more sub-problems of the same or related type, until these become simple enough to be solved directly.
  - Conquer: independently solve each sub-problem.
  - Combine (**for divide and conquer**): the solutions to the sub-problems are combined to give a solution to the original problem.

- E.g.: (1) Binary search (2) Merge sort (3) Greatest common divisor, etc…

Examples

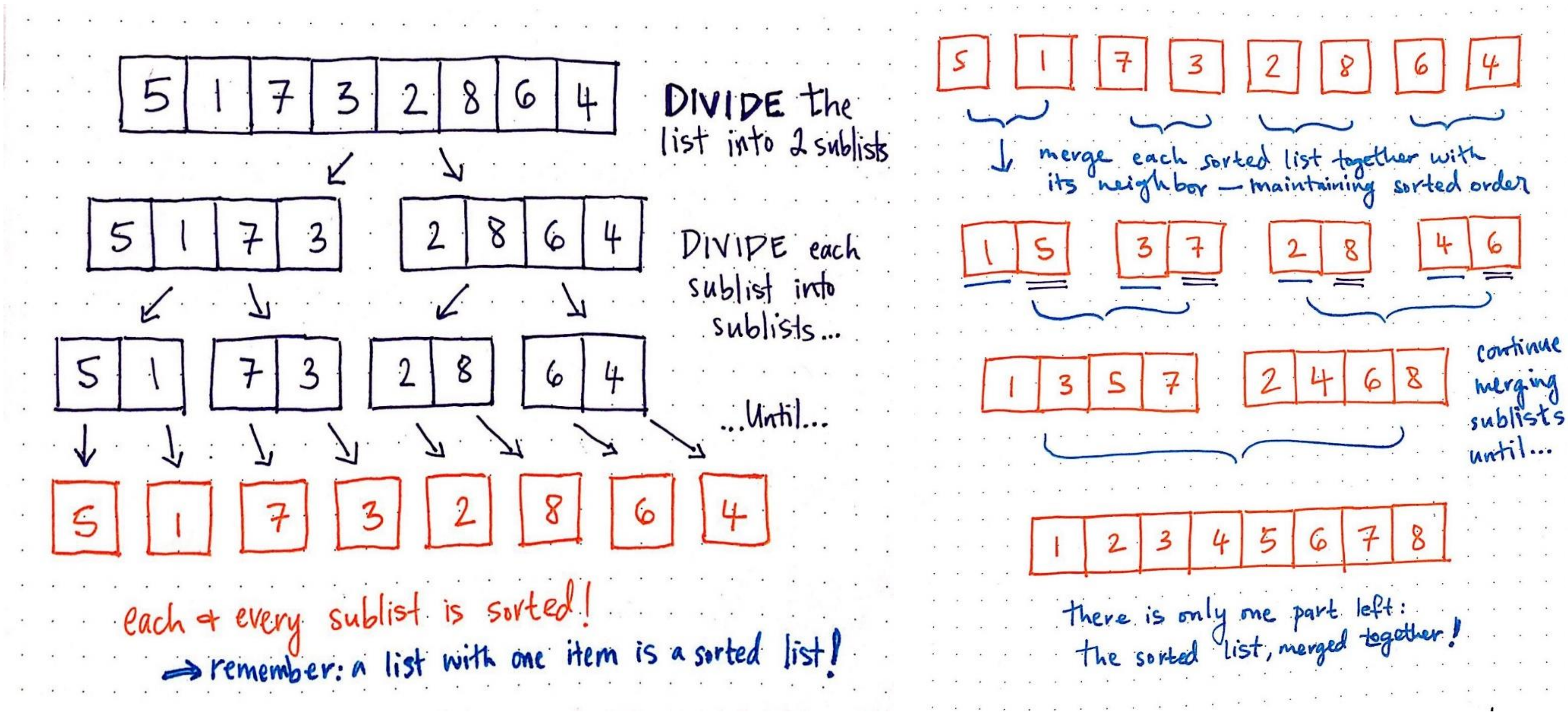
- Binary search: decrease and conquer algorithm



Ref: https://www.topcoder.com/blog/binary-stride-a-variant-on-binary-search/

Examples

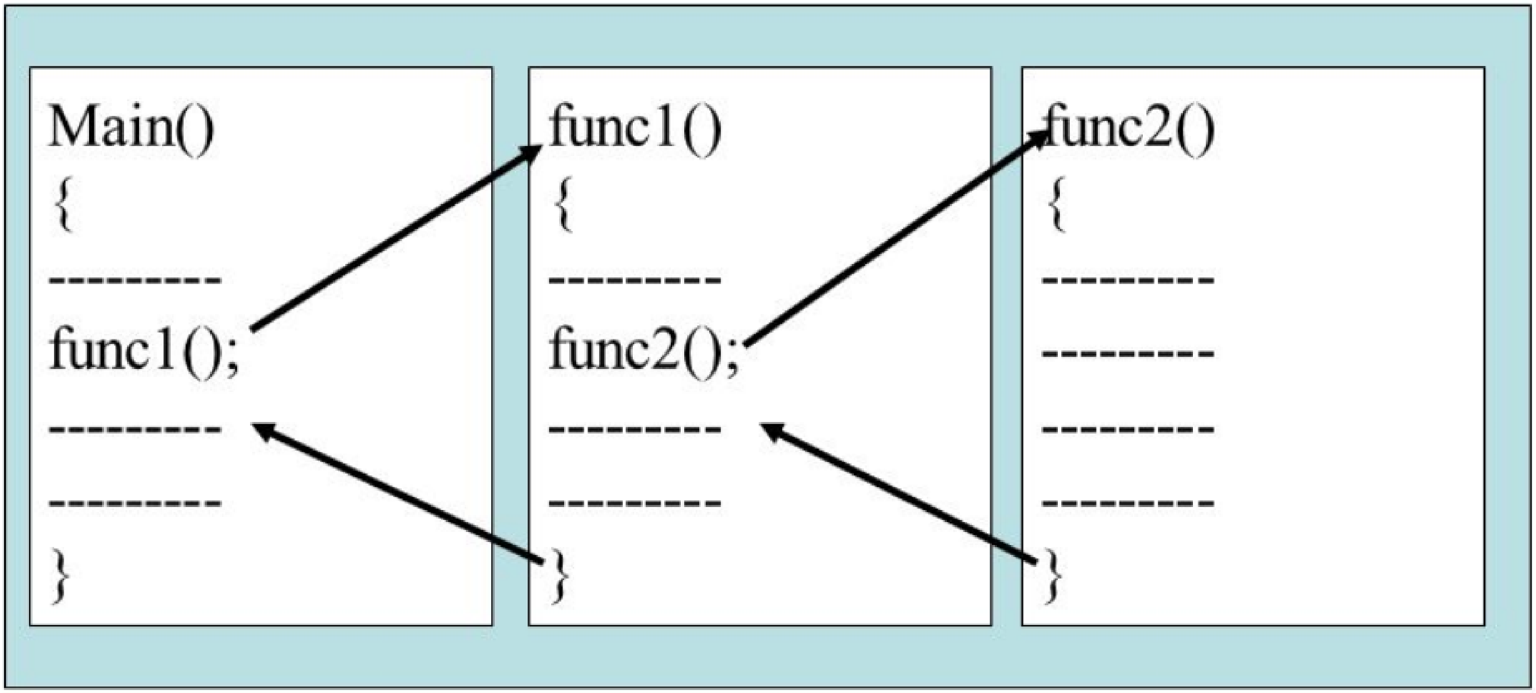
- Merge sort: divide and conquer algorithm



Ref: https://medium.com/basecs/making-sense-of-merge-sort-part-1-49649a143478

Stack Frame Revisited

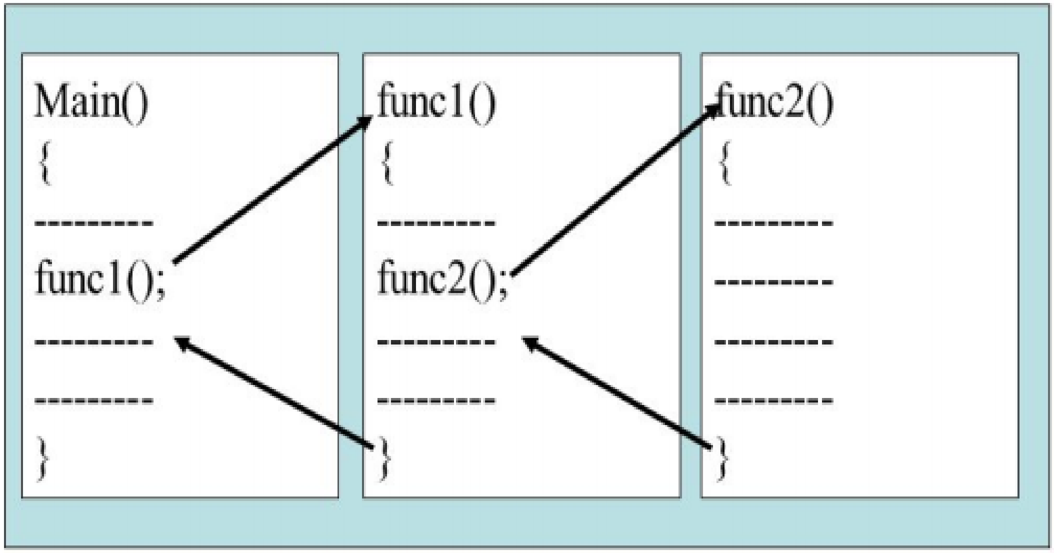
- Think about we want to implement following function:

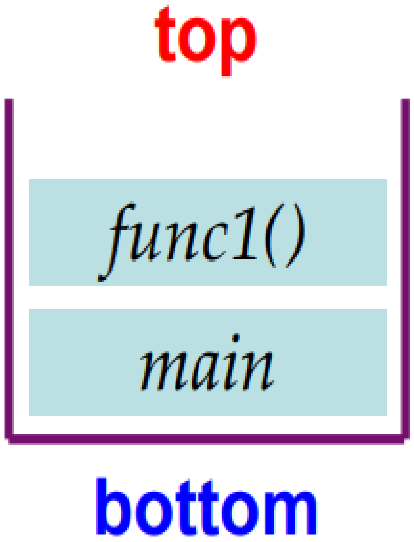


- Observation: the return address needed for the first return is the last one generated in the nested calls.
  - Last-in–first-out (LIFO) order -> Stack

Stack Frame Revisited

- Think about we want to implement following function:





- Observation: the return address needed for the first return is the last one generated in the nested calls.
  - Last-in–first-out (LIFO) order -> Stack

Stack Frame Revisited

- Processor **stack** is useful to store subroutine linkage

Main program

```
                 :
2000             PUSH    PARAM2            Place parameters on stack.
2004             PUSH    offset PARAM1
2008             CALL    SUB1
2012             POP     [RESULT]          Store result.
2016             ADD     ESP, 4            Restore stack level.
2020             next instruction
                 :
```

First subroutine

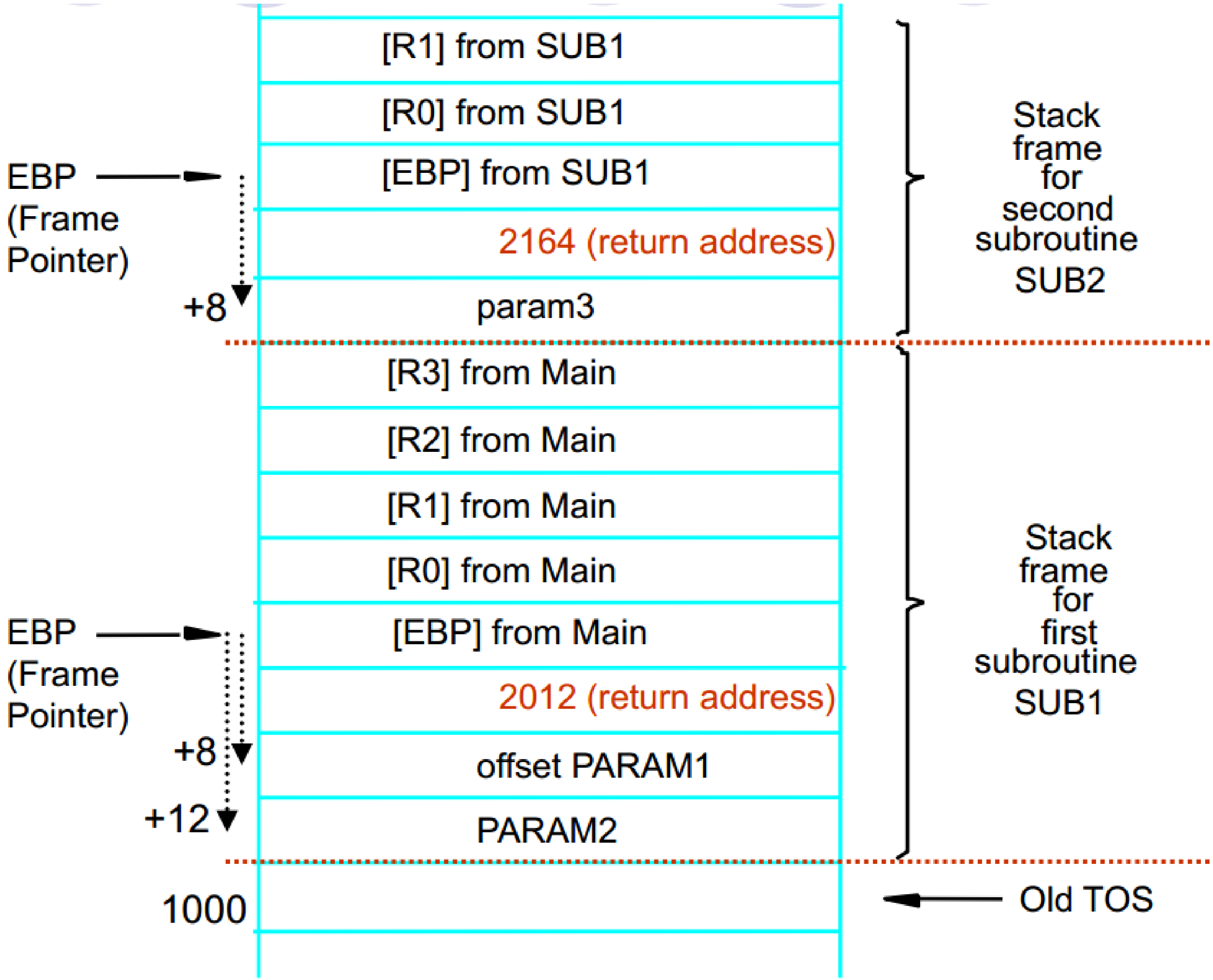
```
2100   SUB1:     PUSH    EBP               Save frame pointer register
2104             MOV     EBP, ESP          Load the frame pointer
2108             PUSH    R0, R1, R2, R3    Save registers (Abbreviated!)
...              MOV     R0, 8[EBP]        Get first parameter.
                 MOV     R1, 12[EBP]       Get second parameter.
                 :
                 push    PARAM3            Place a parameter on stack.
2160             CALL    SUB2
2164             POP     R2                Pop SUB2 result into R2.
                 :
       P1:       MOV     8[EBP], R3        Place answer on stack.
                 POP     R3, R2, R1, R0
                 POP     EBP
                 RET
```

Second subroutine

```
3000   SUB2:     PUSH    EBP               Save frame pointer register
                 MOV     EBP, ESP          Load the frame pointer

                 PUSH    R0, R1            Save registers (Abbreviated!)
                 MOV     R0, 8[EBP]        Get the parameter.
                 :

                 MOV     8[EBP], R1        Place SUB2 result on stack.
                 POP     R1, R0            Restore registers (Abbreviated!)

                 POP     EBP               Restore frame pointer register.
                 RET                       Return to SUB1.
```

Standard form!

Stack Frame Revisited



Stack Frame Revisited

- Conservation of stack level:
  - The major concern is the value of ESP, it is always modified by PUSH and POP instructions.

- Preservation of registers' contents
  - Save and restore the registers' contents before and after they are used for other purposes
  - E.g. in a procedure call

Greatest Common Divisor

- In mathematics, the greatest common divisor (GCD) of two or more integers, when at least one of them is not zero, is the largest positive integer that divides the numbers without a remainder.

- E.g., GCD(60, 36) = 12.

- You can definitely enumerate all divisors of each number and then pick up the largest common one, but it’s not efficient enough.

Euclid's Algorithm

- A simple way to find GCD is to factorize both numbers and multiply common factors.

$$36 = 2 \times 2 \times 3 \times 3$$
$$60 = 2 \times 2 \times 3 \times 5$$

$$\begin{aligned} \text{GCD} &= \text{Multiplication of common factors} \\ &= 2 \times 2 \times 3 \\ &= 12 \end{aligned}$$

- Recursive algorithm:
  - gcd(a, 0) = a
  - gcd(a, b) = gcd(b, a mod b)
  - E.g., gcd(48, 18) = gcd(18,12) = gcd(12,6) = gcd(6,0) = 6
- Question: what is the implementation in C/C++?

Euclid's Algorithm

- Euclid's Algorithm in C/C++

```
int GreatestCommonDivisor(int a, int b)
{
    if (a < b)
    {
        int temp = a;
        a = b;
        b = temp;
    }
    if (b == 0)
        return a;
    else
        return GreatestCommonDivisor(b, a % b);
}
```

- 4 parts: (1) main func of gcd (2) func of check order (3) func of divide recursively (4) how to end

Euclid's Algorithm in MASM

- Complete the Euclid's algorithm in MASM:

- Hints:
  - Complete (2) (3) (4) as separate functions
  - for "div src"
  - EAX = EDX:EAX / src
  - EDX = EDX:EAX % src

```
gcd proc
    push ebp
    mov ebp, esp
    push ebx
    push eax
    mov eax, 8[ebp]
    mov ebx, 12[ebp]

    ; Fill here!
    ; Put the result in ecx

return:
    pop eax
    pop ebx
    pop ebp
    ret
gcd endp
```

Euclid's Algorithm in MASM

```
gcd proc
    push ebp
    mov ebp, esp
    push ebx
    push eax
    mov eax, 8[ebp]
    mov ebx, 12[ebp]

order:
    cmp eax, ebx
    jge divide
    mov ecx, eax    ; a < b
    mov eax, ebx
    mov ebx, ecx

divide:             ; a >= b
    cmp ebx, 0
    je base
    xor edx, edx
    idiv ebx        ; edx = a % b
    push ebx
    push edx
    call gcd
    add esp, 8
    jmp return

base:
    mov ecx, eax

return:
    pop eax
    pop ebx
    pop ebp
    ret
gcd endp
```

Euclid's Algorithm in MASM

- Actually there is a direct implementation of GCD, why still using subroutine?

- What if you want to find GCD of 4 integers? Is that easy to modify with this implementation? NO!

```
start:
    invoke crt_printf, addr PrintFormat1
    invoke crt_scanf, addr ScanFormat, addr Int1
    invoke crt_printf, addr PrintFormat2
    invoke crt_scanf, addr ScanFormat, addr Int2
    mov eax, Int1
    mov ebx, Int2

order:
    cmp eax, ebx
    jge divide
    mov ecx, eax    ; a < b
    mov eax, ebx
    mov ebx, ecx

divide:             ; a >= b
    cmp ebx, 0
    je output
    xor edx, edx
    idiv ebx        ; edx = a % b
    mov eax, ebx    ; a = b
    mov ebx, edx    ; b = a % b
    jmp order

output:

    invoke crt_printf, addr PrintFormat3, Int1, Int2, eax
    invoke ExitProcess, NULL
end start
```

Euclid's Algorithm in MASM

- Benefits of subroutine/function again:
  - Divide and conquer
  - Reuse codes
  - Make variable namespace clean

- Exercise: use the “subroutine” we just define to get the GCD of 4 integers.

- GCD(a, b, c, d) = GCD(a, GCD(b, GCD(c, d)))

Summary

- Subroutine Revisited
- Stack Frame
- Greatest Common Divisor