Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

Department of Computer Science

Computer Architecture II
CSU34021

Tutorial 1: Solutions
Intel's 32-bit Assembly with C/C++

Syed Asad Alam

Document History

Rev.	Date	Comment	Author
1.0	25-11-2020	Solution released	SAA

1 Learning Outcomes

This lab satisfies the following learning outcomes of the course:

LO1 Write simple IA32 and x64 assembly language functions

LO2 Explain the IA32 and x64 procedure calling conventions

LO3 Write programs that mix C/C++ and IA32 or x64 assembly language functions

2 Exercises

2.1 Program 1

The following procedure evaluates the following polynomial:

$$x^2 + x + 1$$

```
int poly(int arg)
{
  int res;
  res = pow(arg,2);
  res = res + arg + 1;
  return(res);
}

int pow(int arg0, int arg1)
{
  int result,i;

  result = 1;
  for (i=1; i<=arg1;i++)
    result = result*arg0;
  return(result);
}
```

The corresponding assembly code is:

```
        public poly
poly:
        ;; Establishing the stack frame through the prologue
        push ebp
        mov ebp, esp

        ;; Main function body

        ;; Retrieving the argument to poly
        mov eax, [ebp+8]

        ;; Creating space for the result from the procedure pow

        push 2                  ; first argument to 'pow'
        push eax                ; second argumnent to 'pow'

        ;; Calling the procedure: 'pow'
        call pow

        add esp, 8              ; clearing the arguments from the stack

        ;; Return value already in eax, so adding the argument which is preserved in the
        ;; stack relative to ebp
        add eax, [ebp+8]        ; eax = eax + argument
        inc eax                 ; eax = eax + 1

        ;; Epilogue, dismantling the stack frame
```

```
        mov esp, ebp
        pop ebp

        ;; Returning from the function
        ret

pow:
        ;; Establishing the stack frame through the prologue
        push ebp
        mov ebp, esp

        ;; Main function body

        ;; Need to use ebx for storing the second argument as edx will be used by the imul instruction
        push ebx                ;callee preserved register
        mov eax, 1
        mov ecx, [ebp+12]       ;first argument retrieved (from the right)
        mov ebx, [ebp+8]        ;second argument retrieved (from the right)

L1:     imul ebx                ;{edx,eax} = eax*ebx, we are only interested in the lower 32-bit result
        loop L1                 ;ecx used as a loop counter

        ;; Retrieving the value of ebx
        pop ebx

        ;; Epilogue, dismantling the stack frame
        mov esp, ebp
        pop ebp

        ;; Returning from the function
        ret
```

2.2 Program 2

The following procedure finds all multiples of K that are less than N by setting the corresponding of a N-sized array element to 1 whenever a multiple is found.

```
void multiple_k(uint16_t N, uint16_t K, uint16_t* array)
{
  for (uint16_t i = 0; i < N; ++i)
    {
      if((i+1)%K == 0)
        {
          array[i] = 1;
        }
      else
        {
          array[i] = 0;
        }
    }
}
```

```
        public multiple_k_asm
multiple_k_asm:

        ;; Establishing the stack frame through the prologue
        push ebp
        mov ebp, esp

        ;; Main function body

        ;; Need to use ebx for storing the second argument as edx will be used by the idiv instruction
        push ebx                        ; callee preserved register
        push esi                        ; callee preserved register

        ;; Creating space on stack for local variables
        sub esp, 2

        ;; Retrieving the arguments from the stack
        mov esi, [ebp+16]               ; address of the stack
        ;; 16-bit integer K, stored on stack as 32-bits, only least significant 16 bits needed
        mov ebx, [ebp+12]
```

```
        ;; 16-bit integer N, stored on stack as 32-bits, only least significant 16 bits needed
        mov ecx, [ebp+8]
        mov ax, 1                       ; Initializing EAX with '1'

L2:     cwd                             ; sign-extend ax into dx:ax
        mov word ptr [esp], ax
        idiv bx                         ; (i+1)%K, i+1 stored in dx:ax and K in bx
        test dx, dx                     ; comparing if remainder is zero
        jnz set_0                       ; if remainder not zero, jump to set array index to 0
        mov WORD PTR [esi], 1           ; else set the array element to '1' if remainder zero
        jmp cont                        ; skipping the set_0 part
set_0:  mov WORD PTR [esi], 0           ; setting array index to '0'
cont:   add esi, TYPE WORD              ; increment the array pointer by 2 bytes
        mov ax, word ptr [esp]          ; retrieve the value of ax from stack
        inc ax                          ; incrementing eax which acts as a second counter
        loop L2

        ;; Retrieving the value of ebx and esi
        pop esi
        pop ebx

        ;; Epilogue, dismantling the stack frame
        mov esp, ebp
        pop ebp

        ;; Returning from the function
        ret
```

2.3 Program 3

The following procedure calculates the factorial of a number N using recursion:

```
int factorial(int N)
{
  if (N==0)
    return 1;
  else
    return N*factorial(N-1);
}
```

```
        public factorial
factorial:
        ;; Establishing the stack frame through the prologue
        push ebp
        mov ebp, esp

        ;; Main function body
        ;; Retreiving the argument
        mov ecx, [esp+8]

        ;; Comparing with zero
        test ecx, ecx           ; checking if ecx==0
        mov eax, 1              ; set eax to '1'
        jz ret_f                ; return '1' if N == 0
        dec ecx                 ; decrement eax
        push ecx                ; push eax onto stack as argument to a recursive call
        call factorial          ; calling the function recursively
        add esp, 4              ; clearing the arguments from the stack
        imul DWORD PTR [esp+8]  ; eax = eax*N, eax contains the return value

        ret_f:  ;; Epilogue, dismantling the stack frame
        mov esp, ebp
        pop ebp

        ;; Returning from the function
        ret
```

2.4 Stack Diagram