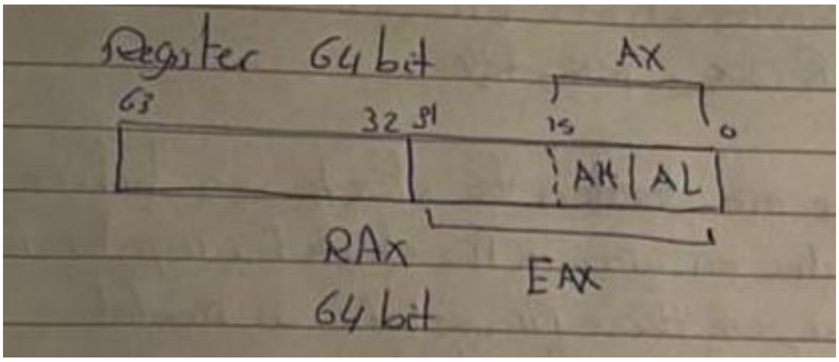
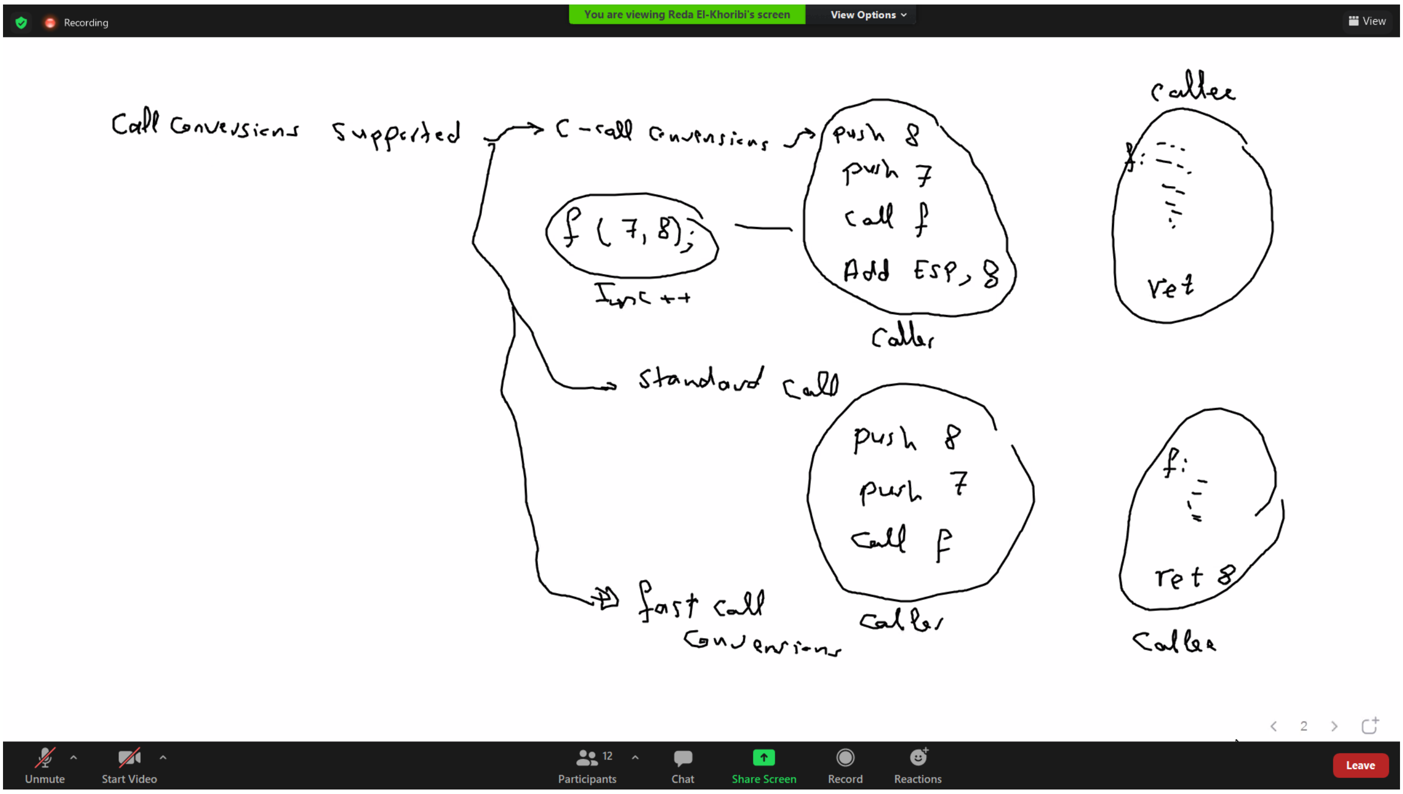
Computer Architecture - Lecture 6

- Intel x64:
    - Registers:
        - RAX, RBX, RCX, RDX, RSI, RDI, RSP, RBP, R8, R9, R10, R11, R12, R13, R14, R15
        - R8 (64 bits)
        - R8w (16 bits)
        - R8d (32 bits)
        - R8b (8 bits)



- Call conversions supported:
    - C-call conversions
    - Standard call
    - Fast call conversion



- In fast call conversion, first integer parameters are stored in: RCX, RDX, R8,R9, and the rest of parameters are stored in the stack as before.
- The callee(function that I call) allocates an area in the stack for the register parameters to save the values of these parameters as needed.
- The caller then emptily the stack using *ret n;*
- Return values of functions are in RAX.

- MIPS ISA:
    - MIPS is a RISC machine..
    - Basic features:
        - Few instructions, all orthogonal (no overlap of functionality).
        - Fixed length of instruction code (in MIPS 32 bits all instruction takes 32 bit = 4 bytes). Why? To facilitate the fetch operation (by adding 4 to PC at the end of fetch).
    - Simplified logic of instruction decoding (structure format).
    - Execution unit is simple → permitting multi-core processors.
    - Registers: 32 registers, each of 32-bit length.
    - Register name: $0, $1, $2,… $31.

      R0, R1, R2,… R31.

      $0 → $zero (always 0)
      $2, $3 → $v0, $v1
      $4, $5, $6, $7 → $a0, $a1, $a2, $a3
      $8 to $15, $24, $25 → temporary variable
      $t0 to $t9
      $16 to $23 → $s0 to $s7, $31→$ra (return address)
- Instructions:
    - All instruction except the load & store do not use memory operands (either registers or immediate)
    - Ex: The intel instruction

      Mov AX, [BX]

      Corresponds to the instruction.

      lw(load word) $1, ($2)

      loads the word with address in $2 into $1.

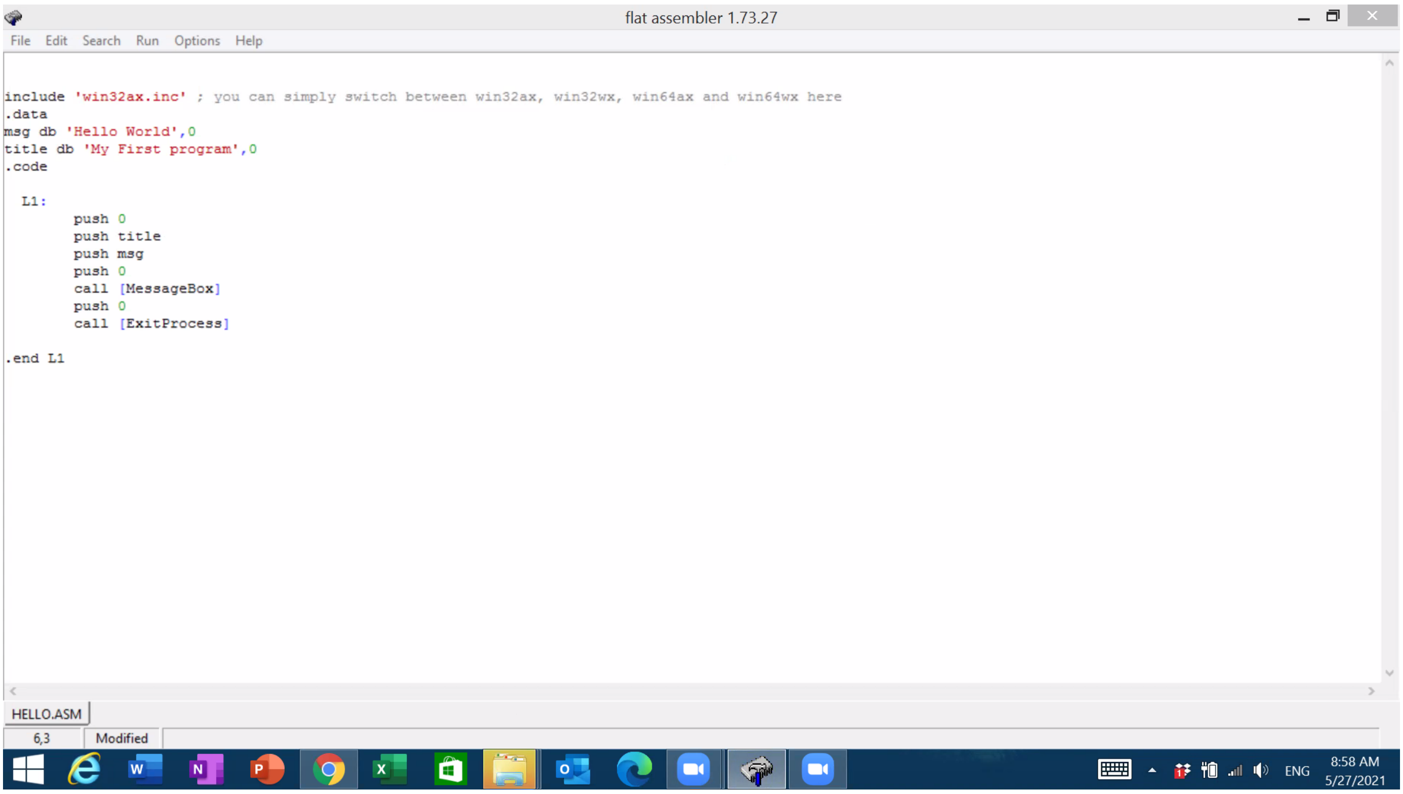
    - The intel instruction
        - Add EAX, [EBX] (SISC instruction)
        - Corresponds to → lw $3, ($2)
        - Add $1(destination), $1(source), $3(target)
- Instruction types & format:
    - Instructions
        - R format (all operands are registers)
        - I format (one operand is immediate)
    - Ex:
        - Odd $1, $2, $3 → adds registers $2 + $3. (R format)

          Store result in $1.
        - Add I $1, $2, 100 → adds register $2 + 100. (L format)

Stores in $1.

        - Unsigned addition: addu $1, $2, $3 (uses unsigned values)
        - Operations → arithmetic: add, sub, mul, divide.
        - Mul $1, $2, $3 → $1 = $2 * $3 (without overflow)
        - Mult $2, $3 → $hi:$lo = $2 * $3
        - Div $2, $3 → remainder stored in $hi, quotient stored in $lo
        - Logic operations: and, or (andi, ori)
        - Ex:
            - And $1, $2, $3
            - $1 = $2 ^ $3
        - Shift: sll, srl
        - Ex:
            - Sll $1, $2, 3 (shifts $2 to left [3 bits])
- Data Transfer: note to transfer from register-to-register use add with zero.
- Intel
    - Mov EAX, EBX
- IN MIPS
    - Add $1, $2, $0
- Load word: (lw), store word (sw)
    - Ex:
        - Lw $1, 50 ($2)
            - Loads the word stored in address $2 + 50 in $1
        - Sw $1, 50 ($2)
            - stores $1 in memory word with address $2 +50
- lui: load upper immediate
    - lui $1, 70 → loads 70 in the upper 16 bits of $1

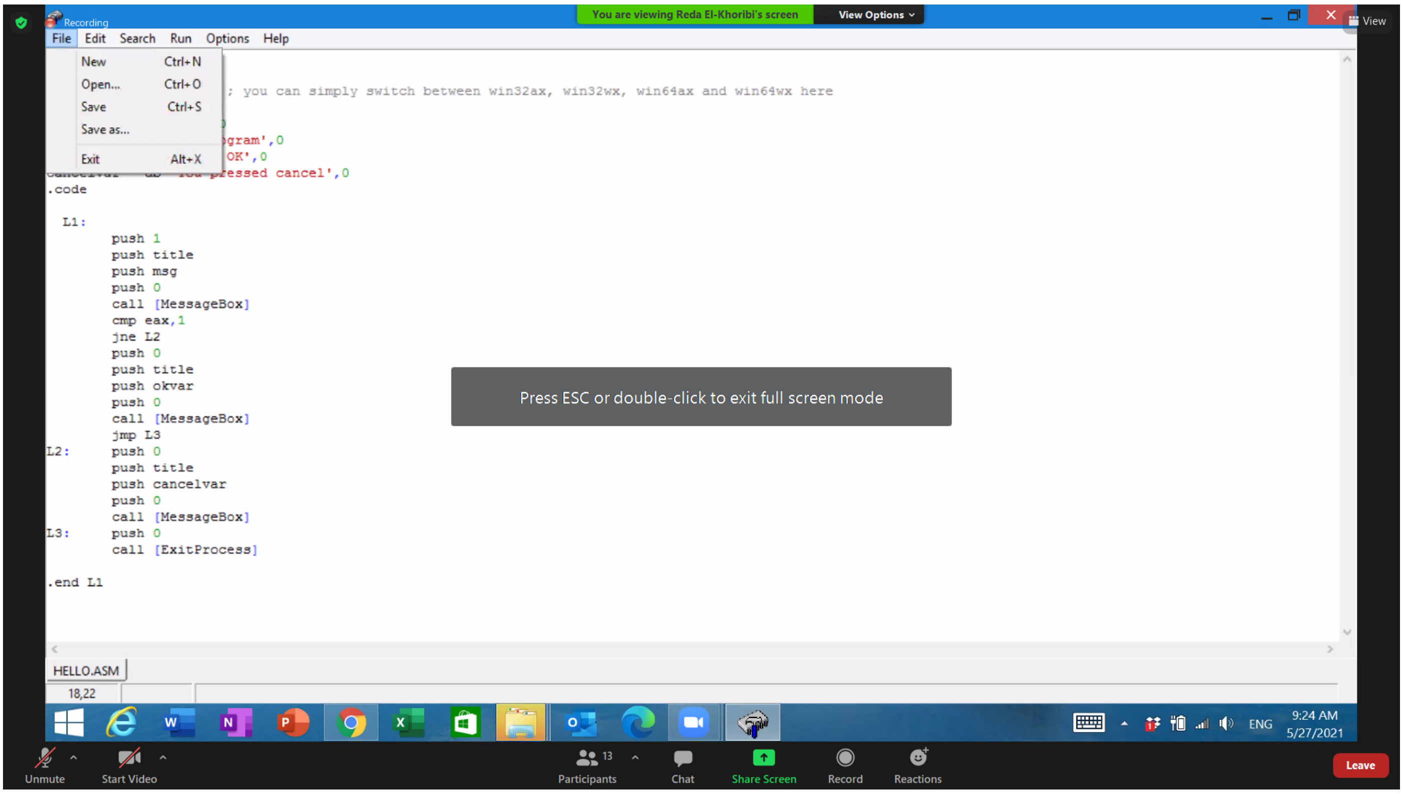
    - move from hi
        - mfhi $1: $1 ← $hi
    - move from lo
        - mflo $1: $1 ←$lo
- Branch instructions beq, bne, bgt, bge
    - Ex: beq $1, $2, 100 (if $1 == $2 then jump 100 instruction forward [400 bytes] [PC + 400]) endif



```
include 'win32ax.inc' ; you can simply switch between win32ax, win32wx, win64ax and win64wx here
.data
msg db 'Hello World',0
title db 'My First program',0
.code

  L1:
        push 0
        push title
        push msg
        push 0
        call [MessageBox]
        push 0
        call [ExitProcess]

.end L1
```



```
                          ; you can simply switch between win32ax, win32wx, win64ax and win64wx here

                         )
                         gram',0
                         OK',0
cancelvar   db  'You pressed cancel',0
.code

  L1:
        push 1
        push title
        push msg
        push 0
        call [MessageBox]
        cmp eax,1
        jne L2
        push 0
        push title
        push okvar
        push 0
        call [MessageBox]
        jmp L3
L2:     push 0
        push title
        push cancelvar
        push 0
        call [MessageBox]
L3:     push 0
        call [ExitProcess]

.end L1
```