**Von Neumann architecture** - stored-program computer where instruction fetch and a data operation cannot occur at the same time because they share a common bus. **linker** - combines individual files created by an assembler into a single executable program.

**assembler** - program that converts source-code programs from assembly language into machine language. **instruction**: executed at runtime

**directive**: a set of assembler instructions that are gathered at assembly time. **IEC cycle**: fetch, decode, execute

**CPU**: clock, registers, control unit, arithmetic logic unit **buses connected to CPU**: data, address, control **wait state**: time delay caused by speed differences CPU, the system bus, and memory **one to one:** assembly language and machine language

**Carry**: unsigned integer overflow **Overflow**: signed integer overflow **Zero**: operation produced zero **Sign**: operation negative (first bit is 1)

**Parity**: even number of 1 bits in least S byte **Aux Carry**: 1 bit carries out of position 3 in LSB

**CISC**: individual instruction might be broken down into several instructions behind the scenes, which makes it more powerful but slower.

**RISC** each instruction is just one step

**ReadChar**: reads single char to AL. **WriteChar**: writes single character to the console. Pass the character (or its ASCII code) in AL

**ReadDec**: reads 32-bit unsigned integer into EAX. **WriteDec:** writes unsigned integer to console with no leading zeros. Pass integer in EAX

**ReadInt**: reads 32-bit signed integer into EAX. **WriteInt:** writes signed integer to console with no leading zeros. Pass integer in EAX

**ReadString**: reads string from keyboard, until Enter key pressed. Pass the offset of a buffer in EDX and set ECX to the maximum number of characters the user can enter, plus 1. returns the count of the number of characters typed by the user in EAX.

**WriteString**: writes a null-terminated string to the console window. Pass the string’s offset in EDX.

**CALL**: pushes its return address on the stack and copies the called procedure’s address into the instruction pointer.

**ESP** always points to the last value to be added to, or pushed on, the top of stack.

**Single Precision x86**: 1 bit sign, 8 bits exponent, 23 bits significand **Double Precision x86**: 1 sign, 11 e, 52 s **Double ext**. 80 bits

**args**: values passed to a subroutine **parameters**: received by subroutine **Stack paras**: compat w/ high-level languages, reduce code clutter

**stack frame** (activation record): area of the stack set aside for a procedure’s return address, passed parameters, local variables, and saved registers. The stack frame is created when the running program begins to execute a procedure: **1**. arguments pushed on stack; **2**. procedure called; **3**. EBP pushed on stack; **4**. EBP set to ESP; **5**. if local variables, ESP decremented to make space; **6**. If registers save, pushed onto stack

**INC/DEC**: reg/mem, add 1 and subtract 1 from a register or memory operand. Does not affect the Carry flag.

**ADD**: dest, source. Source is unchanged, sum is stored in the dest **SUB**: dest, source. Source is unchanged, diff is stored in the dest

**MOV:** Source is unchanged, copied to dest Multiplicand Multiplier Product

**MUL:** reg/mem8/16/32. Product is twice their size in bits. AL|AX|EAX reg/mem8|16|32 AX|DX:AX|EDX:EAX

**IMUL**: (signed). Single operand = same as above, Dividend Divisor Quotient Remainder

**DIV** reg/mem8/16/32. AX|DX:AX|EDX:EAX reg/mem8|16|32 AL|AX|EAX AH|DX|EDX

**Signed division**: need to sign extend: CBW (convert byte to word), CWD (convert word to doublew), CDQ (convert doublew to quadw)

**PUSH:** first decrements ESP, then copies source operand into stack. 16-bit operand decrements by 2, 32-bit by 4.

**POP:** first copies contents of the stack element pointed to by ESP into a 16- or 32-bit destination operand and then increments ESP (2 or 4).

**CALL**: address following the call is pushed to stack; address of proc is loaded into EIP; All instructions in proc execute up to RET.

**RET:** the value in the stack pointed to by ESP is popped into EIP

**TITLE** [text]: Defines the program listing title. ***label* PROC/*label* ENDP**  Marks start/end of a procedure block called *label*

Within a procedure definition (PROC), **LOCAL** creates stack-based variables that exist for the duration of the procedure.

**Constant** **declaration**: MY\_NAME EQU <”Andrew W”,0>

0xD6 0x85 0x41 0xA0 little endian, is what in 32-bit signed? → 10100000010000011000010111010110 – starts with 1, so it’s negative

01011111101111100111101000101010 – flip, +1, → -1606318634

XOR

1001

0011

1010