WEEK 1

ALU = Arithmetic/Logic Unit

CACHE = an area of fast temporary storage

IP = Instruction Pointer

IR = Instruction Register

Instruction Execution Cycle

1. Fetch next instruction
2. Increment IP to point to next inst
3. Decode instruction in IR
4. If instr requires mem access
   1. Determine Mem Add
   2. Fetch operand from memory into register
5. Execute micro-program for instr
6. Go to step 1

Protected Mode – 4 GB available

Real-address mode – 1 MB

wait state – time delay due to differences between the CPU, system bus, etc.

3 types of buses – Data, Address, Control

Parts of instruction from left to right:

* Label, mnemonic, operand, comment

Registers:

EAX and EDX for integer and div instructs

ECX – Counter

ESP – Reference System Stack

ESI, EDI – Extended index registers

Typical Register Diagram

|  |  |  |
| --- | --- | --- |
|  | AH | AL |
| AX | |
| EAX | | |

Data Types:

|  |  |
| --- | --- |
| Type | Used For |
| BYTE | Character, string, 1-byte int |
| WORD | 2 byte int, address |
| SWORD | 2 byte signed int |
| DWORD | 4 byte unsigned int, addres |
| SDWORD | 4 byte signed int |
| FWORD | 6-byte int |
| QWORD | 8 byte int |
| TBYTE | 10-byte int |
| REAL4 | 4-byte floating-point |
| REAL8 | 8-byte floating-point |
| REAL10 | 10-byte floating-point |

Declare an array as follows:

someBytes WORD 42 DUP(0)

It means they are initialized to 0.

Integer Information:

* A signed integers stores the sign in the most significant bit.
* The integer range of ASCII codes is 0 to 127
* 32 bit signed integer range:
* 231 – 1 to -231

MASM Instructions:

**mul**

* + implied operand in EAX
  + results in EDX:EAX

mov eax, 10

mov ebx, 12

mul ebx

**div**

* + implied operand in EDX:EAX
  + extend EAX into EDX before division
  + quotient is in EAX
  + remainder is in EDX

mov eax, 100

cdq ; extend sign into edx

mov ebx, 9

div ebx ;quotient is in eax

;remainder is in edx

**mov**

* + immediate move to DS not OK
  + size mismatches
  + EIP cannot be destination
  + Immediate value cannot be dest
  + Memory to memory not allowed

Irvine Library:

Clrscr – Clear the screen

* + Pre: none
  + Post: screen cleared and cursor at upper left

Crlf – New line

* + Pre: none
  + Post: cursor is at beg of next line

ReadInt – Reads an integer from keyboard, terminated by Enter key

* + Pre: none
  + Post: value entered is in EAX

ReadString – Reads a string from keyboard, terminated by the Enter key

* + Pre: OFFSET of memory destination in EDX, size of memory destination in ECX
  + Post: String entered is in memory, Length of string entered is in EAX

WriteInt, WriteDec – Writes an integer to the screen

* + Pre: value in EAX
  + Post: value displayed,
  + WriteInt displays +/-

WriteString – Writes a null-terminated string to the screen

* + Pre: OFFSET of memory location in EDX
  + Post: string displayed

WEEK 2

JMP = Jump to Label

CMP = compare left to right

cmp eax, ebx

Equality Comparisons:

|  |  |
| --- | --- |
| Mnemonic | Description |
| JE | Jmp if equal |
| JNE | Jmp if not equal |
| JCXZ | Jmp if CX = 0 |
| JECXZ | Jmp if ECX = 0 |
| JRCXZ | Jmp if RCX = 0 |

Unsigned Comparisons:

|  |  |
| --- | --- |
| Mnemonic | Description |
| JA | Jump if Above |
| JNBE | Jmp if not below or equal |
| JAE | Jmp if Above or Equal |
| JNB | Jmp if not below |
| JB | Jmp if below |
| JNAE | Jmp if not above or equal |
| JBE | Jmp if below or equal |
| JNA | Jmp if not above |

Signed comparisons

|  |  |
| --- | --- |
| Mnemonic | Description |
| JG | Jmp if greater |
| JNLE | not less than or equal |
| JGE | Jmp greater than or equal |
| JNL | Jmp if not less |
| JL | Jmp if less |
| JNGE | not greater than or equal |
| JLE | Jmp if less than or equal |
| JNG | Jmp if not greater |

Loops:

* Pre-test loop (while): check condition before body of loop, including the loop control update.
* Post-test loop (do-while): check condition after body of loop, including the loop control update
* Loop statement automatically decrements ecx, will only jump out if ecx = 0 **AFTER** it is decremented. Therefore, it must be at least 1 to begin with.

Constants:

* Two ways to define a constant, however do both before .data
* PI = 3.1416 or PI EQU <3.1416>
* NAME EQU <”Kevin Lewis”, 0>

$ = Current location in data segment

Two’s Complement

* Change every bit to its opposite then at 1 to the result.

Conversion:

* To binary – Divide by 2 until you get 0, remainders are binary code
* To Hex – Divide by 16 until you get 0, remainders are hex code.

16 Bit sign vs unsigned range:

* Unsigned = 0 - 65535
* Signed = -32768 to 32767

Flags:

Carry (CF)

* + Number is larger than the size of the holder. 16 bit number in an 8 bit reg.
  + Or if a negative number is produced on with an unsigned subtraction.
  + INC instruction does not affect it

Overflow (OF)

* + Sum of two numbers with sign bits off yields a result number with the sign bit on.
  + Sum of two number with the sign bits on yields a result number with the sign bit off (doesn’t care if signed or unsigned)