CH1 The MIPS Architecture

- Assembly Language can open opportunities in the career field of **Embedded Processors**.
- **Architecture** – (Of a computer) is made of: The available(Visible to Programmer) Registers / the Instruction Set / the Memory-addressing Modes / the Data types.
- **Datapath diagram**- Depicts essential components/features of MIPS Architecture.
- **Machine Language**- all Binary to the computer.
- **Operands**- Instructions/Values operated upon.

- <u>Basic Functional</u> <u>Components of MIPS Architecture</u> – Control Unit / Register File / Arithmetic and Logical Unit (ALU) / Program Counter(Pointer) (PC) / Memory / Instruction Register (IR).

- <u>Buses</u> connect everything, except Control Unit.
- **Bus** – set of electrical conducting paths, Binary values are transmitted through.
- **Multiplexer(s) (**or **Data Selector(s))**- Hardware, routes info from more than one source to a destination. If it has 2 input buses, must have a single control wire connected.
- **Control Unit –** Sends a control signal of 0 or 1, in a specific sequence, to select which input bus should be routed to the output. Fetches Instructions from Memory & Executes them.
- **Register –** Storage Component of electronics. (Most hold a 32-bit Binary Number)
- **Register File –** Contains 32 Registers.
- <u>Range of Register Values</u> =decimal values represented by 32 bits = -2,147,483,648(-2^31) TO 2,147,483,647(2^31 -1)
- **Convention** – (The Usual Way) Specifies which Registers are appropriate for certain uses. $_ sign = Register_
- **Macro Instructions** – The Assembler implements these. ($at [Known as *$AssemblerTemporary*) is reserved for this) (Assembly language programmers don't use this Register.)
- Registers $k0 / $k1 are used by the Kernel of the Operating System (OS)
- **Return Values**- Registers 2/3 ($v0 / $v1)
- **Pass Arguments** to Functions– Registers 4,5,6,7 ($a0, $a1, $a2, $a3)
- **Temporary Values –** $t0-$t9 store them. These are typically used for Writing Functions by Programmers.
- **Saved Values –** $s0-$s7 (Non-Modified Values)
- **Stack Pointer -** $sp - points to a segment of memory, called the Stack. Initialized by the OS.
- **Return-Address-** $ra – loaded with return address everytime the machine executes an Instruction to call a Function.
- **Frame Pointer -** $fp – establish a constant reference offset to local Variables / Function Parameters.
- **Arithmetic and Logical Unit (ALU)** – Performs Binary Interger Arithmetic Operations / Binary Logical Operations (AND/OR/NOR/Exclusive OR) a digital logical circuit. Operates depending on Operation Code in the Instruction, fetched from Memory.
- **Program Counter(PC)**(Pointer)- After Instruction is fetched from Memory & Loaded into the IR, the PC is incremented so the CPU will have the address of the next sequential instruction for the next instruction fetch. Register, initialized by OS.
- **Memory –** Large array of location. Binary Info is stored/fetched, 1 Word at a time.
- **Word –** 32-bit Quantity. **Half-Word**= 16-bit value. **Byte**=8-bit value/the smallest value. Words contain 4 Bytes. The Address of the First Byte=Address of the Word.

CH1 The MIPS Architecture

- **Memory Addresses –** Range from 0-4,294,967,295(2^32 -1).
- **Instructions –** All are 32 bits (4 Bytes)
- **Instruction Register (IR) –** 32-bit Register. Holds a copy of most Recent fetched Instructions.
- **Instruction Formats** (3)**-** Register Format / Immediate Format / Jump Format
- **Immediate Format** – perform Arithmetic/Logical operations between **Variable** (Stored in Register File)& **Constant** (Stored in Instruction). Branch Instructions / Load and Store Instructions = this format.
- **Jump Format** – Equivalent of "Go To" Instruction (High-Level Languages)
- **MIPS Instruction set** – Unique Binary codes for each Instruction. EXAMPLE: add $v1, $a1, $s1 PSEUDOCODE: $v1=$a1+$s1
- **Control Structures** – "if"/"then"/"else"
- **Multiply Instruction** – "mult" – Multiplies (2x) 32-bit Binary Value to get a 64-bit Binary Value. Stored into 2 Registers (High/Low). Register High=Upper 32-bits. Register Low=lower 32-bits.
- **Divide Instruction** – "div". Divides 32-bit Value by 32-bit Value. **Quotient**=Register Low. **Remainder**=Register High. Division by 0 = Undefined.
- **Load/Store Architecture** =MIPS Architecture. Only Instructions that access Main Memory are the Load and Store instructions.
- **Addressing Mode (**also **Base Address plus Displacement)** = Only 1 is Implemented in the Hardware.
- **Load Instruction** – accesses a Value from Memory and places a copy of the Value found in Memory in the Register File.
- **Effective Address** =Memory Location, computed, to be accessed by adding together the **Base Address**(Always Content of 1 of the Registers in the Register File) + **Displacement**(Constant Value 8) and Places a Copy of the Value into $s1, for example. EX: $s1 = Mem[$a0+8]. MUST be a # that is a Multiple of 4 because every 32-bit word contains 4x 8-bit Bytes. A Pointer to some location in the Array of Memory.
- <u>Constant Values</u> can range from -32,768(-2^16) to 32,764(2^16 -4)
- <u>Store Word Instruction</u> = sw $s1, 12($a0) == $s1 = Mem[$a0+12]
- **Algorithm**- Step-by-Step Instructions
- **Assembler**(Utility Program) translates the **Mnemonic**(Coding of Programmer) into **Machine Language**(Binary), then stores it in a File on Disk. TO EXECUTE, a **Linking Loader**(Another Utility Program) Loads/Links all necessary Machine Language Modules into Computer's Main Memory, and the PC is loaded with the Address of the 1st Instruction of the Main Program. The Last Instruction executed in the program Returns Control back to the OS(by using a System Call that generates a Software Exception).
- **4 STEPS of OPERATIONS to FETCH/EXECUTE any REGISTER FORMAT INSTRUCTIONS.**
- **Instructions Fetch Phase** – Instruction, Fetched from Memory, Location specified by PC. Is Loaded into IR. PC is incremented by 4.
- **Operand Fetch Phase** – 2x 5-bit codes, Rs and Rt, within Instructions specify which Register File Locations are used to get the 2x 32-bit Source Operands. Decode the Op-Code.
- **Execute Phase** – 2x 32-bit Source Operands are Routed to ALU Inputs(possible Operations performed there, depending on Op-Code in Instructions)
- **Write Back Phase** – Result of Operation, placed into specified(by 5-bit Rd code in IR) Register File. Go to Step 1.