I. Learner Objectives:

At the conclusion of this unit, participants should be able to:

- Examine the functionality of different registers
- Categorize registers
- Identify register sizes
- Explain the purpose of registers

II. Prerequisites:

Before reading this unit, participants should be able to:

- Differentiate between Harvard and von Neumann Architectures
- Explain the fetch/decode/execute instruction cycle
- Summarize the computer architecture and MCU lessons
- Convert between decimal and hexadecimal systems and vice versa

III. Keywords:

Register file, load-store architecture, memory-mapped registers, data registers, address registers, general purpose registers, floating point registers, constant registers, port registers, vector registers, special purpose registers

IV. Instructional Material:

Registers are considered very high-speed storage units or memories available in microprocessors and microcontrollers. Registers are created from flip-flops. Each bit of storage requires one flip-flop.

Most registers are located in the central processing unit (CPU). Generally registers are arranged in an array referred to as a *register file*. Unlike most memory chips that are available to be accessed by the microprocessor or microcontroller which share ports, the register file makes it possible to read from two registers and write into one register simultaneously. Usually the register file entries and the actual registers map one-to-one. The 32 core/general purpose registers that are used by the Microchip PIC32MX460F512L microcontroller are located within a register file.

The number of registers and sizes vary between computers. A 32-bit CPU contains 32-bit wide registers. The Atmel ATmega64L is an 8-bit microcontroller, hence, it has 8-bit wide registers. The Microchip PIC32MX460F512L uses 32-bit registers. These base register sizes determine the size of operands used with instructions as well. For the

PIC32MX460F512L only 5-bits of an instruction is required to access these registers (note: 2^5 = 32 possible registers).

Most computers adopt the load-store architecture. Data is loaded from a larger memory pool into registers. The registers are manipulated through arithmetic, logic, and/or test instructions and the result is stored back into the larger memory pool. The MIPS32 processor uses a load-store architecture, which means only load and store operations may access main internal SRAM memory.

Most registers are included within the CPU. Some registers, however, are located on a separate memory unit. These are considered memory-mapped registers. Memory mapped registers use the same data bus as data memory. Sharing a data bus slows down the load and store operations that affect registers and data. Also, assembly instructions that perform operations on the contents of general purpose registers cannot be applied to memory-mapped registers. Only memory (most data transfer) instructions may be applied directly to memory-mapped registers.

Microcontrollers and microprocessors take of advantage of registers during instruction execution. Registers may be organized into several categories based on scenarios in which they are used. The categories include: data, address, general purpose (GPR) or multipurpose, floating point (FPR), constant, port, vector, and special purpose. Each of these classes is explained in the following paragraphs.

In C programming, these registers are somewhat transparent. We will need to dive into MIPS assembly to understand the exchange of information between registers and data memory.

Data registers store integer values corresponding to data values. The accumulator register found in some simpler processors is an example of a data register. This register is used implicitly to store results of arithmetic operations.

Address registers hold memory addresses that may used to indirectly access a location in memory. In Intel based processors index registers, base index (EBX), destination index (EDI), and source index (ESI), may store address offset values and not address values. The Atmel ATmega64L has 6 registers (R26 – R31) which may store addresses. MIPS uses register *ra* to return the address of the last subroutine call.

General purpose registers are usually the predominant class of register. These registers may be used to store data and addresses as dictated by a program. Many times these registers may be used to

store various sizes of data and addresses. The ranges are generally between 8-bits and 32-bits. Common operations applied to these registers include: arithmetic and logic, data transfer, bit and bit-test, and microcontroller control. The instruction set for the particular microcontroller or microprocessor that is used, describes the types of registers that may be applied to any one particular instruction. The Intel x86 programming model allows for 7 different general purpose registers. The EAX, EBX, ECX, EDX, DBP, EDI, and ESI. The EAX, EBX, ECX, and EDX may be addressed as 8-bit, 16-bit, or 32-bit registers. The DBP, EDI, and ESI registers may be addressed as 16-bit or 32-bit registers. On the other hand the AVR programming model allows for 32 different GPRs. GPRs, R0 – R31, are all 8-bit registers. Registers R26 – R31 may be used as 16-bit address pointers for indirect addressing. R26 and R27 may be accessed as the X-register, R28 and R29 may be addressed as the Y-register, and R30 and R31 may be addressed as the Z-register. The high and low 8-bits of these registers may be accessed using rH and rL, where r represents the name of the register. The high 8-bits are actually the most significant bits and the low 8-bits are the least significant bits. The X, Y, and Z may be used as pointer registers that store the address of other registers or operands. The x86, AVR, and MIPS architecture follow Little Endian notation. Recall Little Endian indicates that the most significant bits of data are stored at larger addressed locations. Further, instructions that operate on the general purpose registers have direct access to them, thus they as a rule execute in 1 clock cycle. Each of the registers has been mapped into the first 32 bytes of data memory. You will read documentation which states the MIPS architecture supports 32 32-bit general purpose registers, this is not completely accurate. As we have already seen *ra* stores an address. In most applications, we can consider only registers v0-v1, a0-a3, t0-t9, and s0-s8 to be general purpose.

Floating point registers store floating point numbers. These registers are involved in floating point arithmetic, such as addition, subtraction, multiplication, and division. The ATmega64L does not support floating point arithmetic and thus does not have any floating point registers. However, the Microchip MIPS32 PIC32MX460F512L supports floating point registers. We will not use these in this course.

Port registers or peripheral registers (also considered Special Function Registers (SFRs) in this case) are used to manipulate pins corresponding to ports. The PIC32MX4 has three base control registers necessary for port operations. These include the tri-state control register (TRISx), PORTx, and latch register (LATx), where x represents available ports A – G. The TRISx register is used to set the data direction of a digital pin. A pin may be set as input or output. If a