

Chapter 1. Introduction

A solution to an engineering problem is based on a set of criteria that judge its success. The same can be said for designing microprocessors. Depending on the application, the best solution can be the fastest, the least expensive, or the smallest. To achieve the best balance that optimizes a design, a number of different architectures and techniques can be used. Even with a powerful design, a processor can be hindered by its slowest component, the memory. Memory performance can be improved using several methods, but it is difficult to answer the question, “What method is best for my application?”

1.0 The 35VEE8 Project Overview

Designing a modern computer processor is a labor intensive task that can span several years. The 35VEE8 project is an opportunity for students to participate in the complete design process as well as gaining in-depth exposure to microprocessor operations. The EE435-436 class sequence follows modern methods of microprocessor design using the same tools and methodologies found in industry. The task of the designer is to build the microprocessor using project specifications while optimizing for area, speed, and cost. The design process begins by examining the required specifications and developing the behavioral model of the microprocessor. In EE435, Computer Organization and Design, a high level design simulates all the behavior the microprocessor but is not limited by hardware constraints such as size and available parts. In EE436, Advanced Digital Design, students work in teams to rework the high level designs to fit within all the hardware constraints. All the steps followed in EE435 and EE436 will be discussed as well as individual work in the team portion.

1.1 Design Methodology

Designing a microprocessor requires an organized approach that facilitates an efficient use of resources and time. A detailed design methodology describes the flow of activities, an order of events, and procedures used to accomplish the task. The entire design organizes into a set of smaller tasks that facilitates a team of people coordinating their activities. Early risks are identified and resolved and progress is more easily measured. All these methods are employed in designing the 35VEE8 processor.

Development of the 35VEE8 occurs over a nine month span beginning at a high level of abstraction spiraling down to the implementation in hardware. This method of reiterating the design process several times while lowering the level of abstraction is called the Spiral Design Methodology. Each iteration results a working design called a Virtual Prototype that mimics the behavior of the microprocessor and the final iteration results in a working device. The EE435 portion of the project produces a complete high level design in one iteration of the spiral cycle. Both major components, the data path and control unit are designed using VHDL (Very High Speed Integrated Circuits Description Language) and Mentor Graphics tools for simulation and layout. Testing determines the correctness of the design.

The focus of the EE436 portion of the project is the implementation of the design onto hardware. Teams of three students work together using the Integrated Product Development Team concept. Using the information gained from the previous cycle, the design team plans and coordinates activities for the next iteration. Tasks are assigned according to the strength and weaknesses of each member while cooperating on problem areas as they arise. The 35VEE8 is constructed from a parts list that includes three

ACTEL FPGA's (Field Programmable Gate Arrays), and a limited number of memory devices. Mentor Graphics tools are used for both the simulation and testing of the lower level design and the placing and routing of the FPGA.

1.2 Improving the Performance of Memory

Microprocessor designers have several methods available to improve memory performance. With the addition of new components or complexity, designers have to know what method has the ability to comply with system requirements yet improve performance. Cache memories are the method most widely used to improve performance but they are limited by the same memory bottleneck that inhibits main memory. Memory controllers are a new concept that exploits the best performance of memories.

Cache memory is duplicate storage buffer that is faster and smaller than main memory. It exploits the memory access patterns of the microprocessor by holding the most commonly used data and instructions. The purpose of the cache is to provide the microprocessor faster access to needed data and instructions.

Memory tends to have slower access time when the stream of requested addresses are in different areas of memory. The purpose of a memory controller is to reorder the requests to memory providing a seamless supply of data to the processor in the original order that was requested.

Chapter 2. Computers

Understanding basic computer components and their operation within a computer is essential to microprocessor design. There are five standard components found in all computers: memory, input, output, data path, and control unit. The data path and control unit comprise the microprocessor which is considered the brains of the computer. The memory stores data and the input and output components communicate with the outside world.

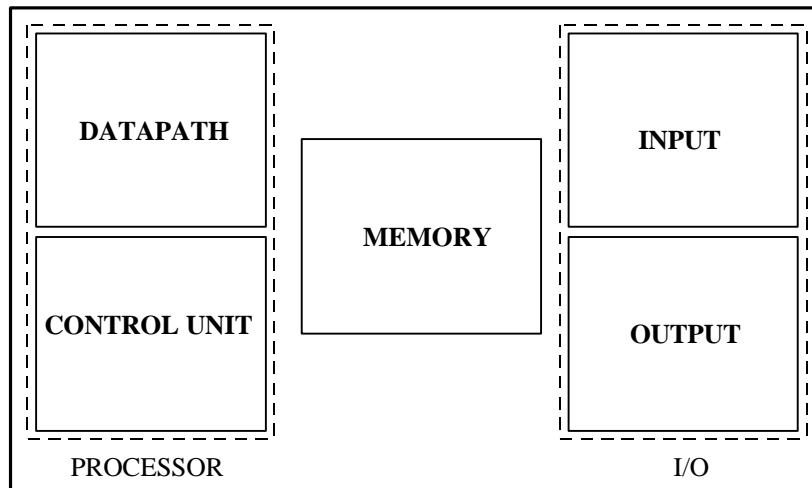


Figure 1: The five basic components in a computer.

A typical operation a could involve taking data from some input device, processing the data, and then sending the data to the outside world through some output device. A computer executes a program by reading a sequence of instructions stored in memory. All instructions and data within a computer are represented in binary form of ones and zeroes packaged into some specified number of bits called a word. A bit is a placeholder for a one or zero and a word represents a group of bits. A user relies on software to provide an interface to the microprocessor so that communication does not have to occur in binary. A microprocessors ability to process data depends on the word size. Standard word sizes

occur in increments of 2^n making word sizes 2, 4, 8, 16, etc. bits. The larger the word size, the more information that can be processed and transferred at once. Current PC technology is up to 32 bits. For the 35VEE8 processor, the word size is 8 bits or 1 byte.

2.0 Memory

The function of computer memory is to store data and instructions. Memory can be further classified as secondary and main memory. Main memory is fast memory that holds active data and programs (information currently being used by the microprocessor). Information is stored in many semiconductor memory cells, each cell holding one bit of information. Word sized groups of cells are accessed when reading or writing information in memory. A unique address is associated with each sequential word location in memory.

The two types of main memory are ROM and RAM. RAM (Random Access Memory) can be used to both read and write information and is generally used to store program data. The user or the microprocessor cannot write to ROM (Read Only Memory) and it generally holds the program and its embedded data. For the 35VEE8, RAM will be used for data storage and ROM's will be used for the storage of programs and its data. Together these memories will make up the microprocessor main memory.

There are several different varieties of main memories available on the market. For the 35VEE8 project, EPROM's and SRAM's are used for the ROM and RAM components. EPROM stands for Erasable Programmable Read Only Memory. It can be erased and then reprogrammed as a whole device but still is not accessible for writing during program execution. SRAM stands for Static Random Access Memory. A constant power supply is needed to maintain the stored information but has greater speed than ROM memories.

Secondary memory holds information that is not accessed as frequently. It is less expensive than main memory but has slower access time compared to main memory devices. Secondary memory devices include hard drives, floppy disks, and tape drives. The 35VEE8 project does not use any secondary memory devices.

2.1 Input and Output

The input component accepts and processes data from an external devices such as keyboard, mouse, etc. making it available to the microprocessor. The output component takes and processes data from the microprocessor and makes it available to external devices such as a monitors, printers, etc. These components are often referred to as the I/O system because they share functionality. The components that make up the I/O subsystem for the 35VEE8 are the I/O controller, UART (Universal Asynchronous Receiver/ Transmitter), a clock, and a driver. The 35VEE8 microprocessor interacts with the I/O system in a similar manner as with memory. Since the memory and I/O systems share the same communication lines (buses), both the memory and I/O systems are able to read information on the communication lines and determine who needs to act. For the I/O system, this means once its determines the information is meant for itself, it takes the address and reads or writes to some location (which is actuality, a external device).

All the I/O components and their arrangement needed for the 35VEE8 project are provided to the design teams. Although the system does not have to constructed, the methods and protocols required for communicating with the I/O system are part of the project specification.

2.2 Communication

Communication between the microprocessor and the I/O and Memory takes place over several group of wires called a bus. Generally computers have three buses that work together to transfer data to and from the microprocessor. Each bus is composed of wires where each wire is able to transmit one bit of information. The address bus is a one-way line that maps into every location in memory. The data bus is a bi-directional line that allows data exchange between the microprocessor, the I/O system and the Memory. The 35VEE8 microprocessor has a sixteen bit address bus that translates into 64K (2^{16} or 65,536) of unique data locations and an eight bit data bus. The control line bus, whose signals are generated by the microprocessor, controls all data transfers by telling the I/O and memory what action is required with the current data and address. Once the data transfer has been accomplished, the I/O and Memory must inform the microprocessor that the transfer has been successful.

2.3 Microprocessor

The microprocessor is the brains of the computer since it controls all the processing and transfer of data. It can be further decomposed into two functional units that each contain storage and combinatory logic. Storage logic is capable of holding data for any required amount of time and combinatory logic produces some new output for some given inputs.

The architecture are those attributes of a system that are visible to a programmer. These include the instruction set, word size, I/O mechanisms, and the methods for addressing memory. Since these attributes are not visible to a programmer, different designs can be used to implement an architecture. The same can be said for the

design of the 35VEE8: its design is made up the five basic computer components but the exact design of each component is up to individual designers.

2.3.1 Data Path

The data path is responsible for the storage and data processing portion of the microprocessor. Inside the data path can be found registers, multiplexers (MUX's), buses, and an arithmetic and logic unit (ALU). A register is a logic device that can store information. The 35VEE8 has both 8 and 16 bit registers that accept data on the rising edge of a clock (positive edge triggered) when enabled. The controlling edge of the clock cycle is time instant that the clock rises from zero to one or falls from one to zero. This means that if a register is holding some information, it must receive an enable signal to change the value of its stored information and the change can occur only at the exact instant of the clock controlling edge.

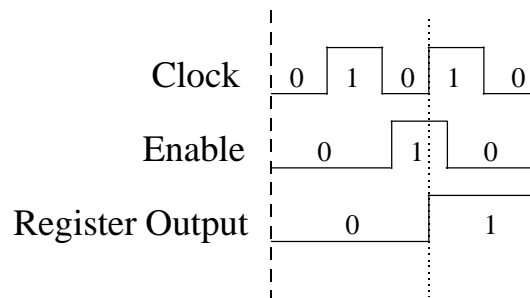


Figure 2: Example of the timing for a positive edge triggered register.

When signals travel through logic devices, there is always some delay until the signal appears on the output, which wasn't shown in figure 2. Delays are a serious consideration for all microprocessor designers especially for registers. Some terms that are used to describe required time intervals for registers are setup and hold time. Setup time is the amount of time that data must be present and stable before the controlling edge

of the clock. Hold time is the time interval that data must remain present and stable after the controlling edge. Simply stated, when new data is provided to a register, changes to that data while the signals are propagating through the register can cause instability.

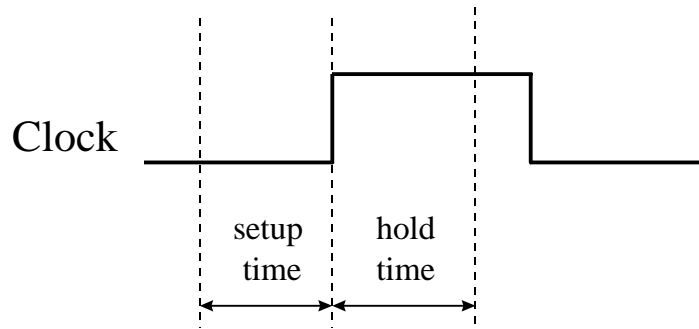


Figure 3: Example of setup and hold time for a positive edge triggered register.

All microprocessors require a set of registers needed to hold data and addresses. One of these is the program counter (PC) which holds the address in memory of the current instruction being executed. It also requires an instruction register (IR) to hold the current instruction. A memory address register (MAR), and a memory data register (MDR) are needed to communicate with the outside world. These hold the address of the current memory location being accessed, and data that is to be written or to be read from memory respectively.

A multiplexer is a logic device with a group of data and control inputs. The control inputs select one of the data inputs and connect it to the output. The number of control lines determines the number of possible inputs. For example, a 2-to-1 MUX has one control line (0 for input 1, 1 for input 2). The 35VEE8 uses both 8-to-1 and 16-to-1 MUX'es that have control lines of three and four bits respectively.

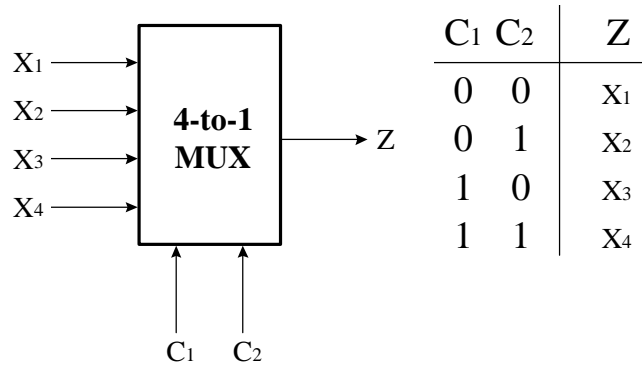


Figure 4: Example of a 4 to 1 MUX and its outputs.

Most processing within any computer takes place within the arithmetic and logic unit. The ALU performs arithmetic and logic operations on both one and two operands. Control lines tell the ALU what operation it is to perform, and there is often a one bit input called a carry-in that factors into certain operations. The ALU generates one resultant and some status bits (carry-out, overflow). The status bits are used by the microprocessor to determine if the operation generated some undesired condition. The ALU of the 35VEE8 can carry out ten different operations on one or two 8 bit operands generating an 8 bit result. It also generates three output bits, carry-out, overflow, and zero on every ALU operation.

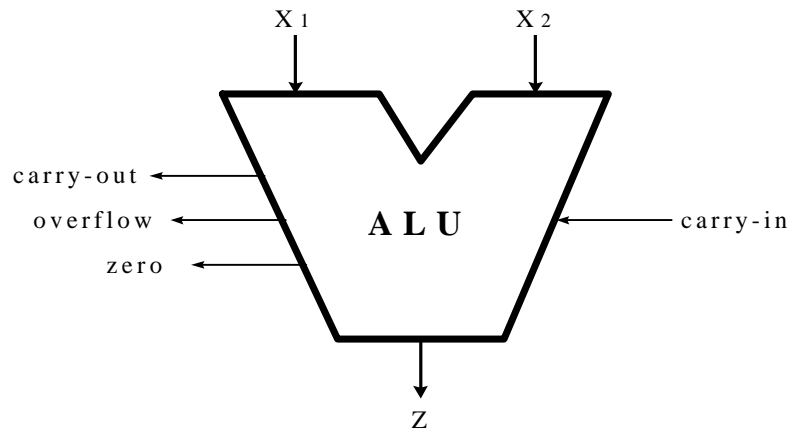


Figure 5: Symbol for the ALU of the 35VEE8

2.3.2 Control Unit

The data path provides all the components needed to store and process data while the control unit directs its operation. It also controls the transfer of data to and from external devices like the memory and I/O systems. Within the data path there are control points for all registers, MUX's, and the ALU. The set of signals that the control unit sends to the data path on every clock cycle is called a micro-instruction. These signals move and process as much of the instruction as possible on each clock cycle. The reason that only one set of signals can be asserted a clock cycle is because data can only be loaded into registers on the controlling edge of the clock and also to prevent cycles (data cycling through the same path). An example micro-instruction would enable a register, let the data through a MUX, process the data in the ALU and then have it present on a bus for some operation on the next clock cycle.

There are two methods of asserting control signals for microprocessors, hard-wired and micro-programmed. The hard-wired design uses logic to determine the control word for every clock cycle. It is generally faster than micro-programmed control but is more complicated and less flexible to change. The micro-programmed method allows easier change to control words since they are stored in a ROM called a micro-store. The speed of the a hard-wired control unit is dictated by the propagation delay for the logic while the speed of the micro-programmed unit is dictated by the access time for the micro-store. The control unit for the 35VEE8 uses the micro-programmed method with two ROM's for the storage of control words. The output of both the ROM's is only 24 bits while the 35VEE8 microprocessor requires some 33 bits to control its operation. When a micro-store has the ability to hold all the controls signals for a microprocessor without

encoding them, it is called horizontal micro-code. The micro-store can release a control word with each bit having an assigned control point. When the micro-store has to highly encode control signals in order to compact the control word, it is called vertical micro-store. The 35VEE8 uses a combination of the two called diagonal micro-code. Some fields within the ROM are not compacted at all while others need decoding at the source to generate all the needed control signals.

To sequence the access of control words in the micro-store as well as handling resets and interrupts, the 35VEE8 control unit uses a state machine. This logic device accepts status inputs from the data path and micro-store to determine what set of control signals to assert on the next clock cycle. Figure 6 shows a simple state machine with one status input. While in any of the states, the state machine will transition to a new state at the controlling edge of the clock if the status bit is equal to one, otherwise it stays in the current state. A state machine also releases output(s) on every controlling clock edge.

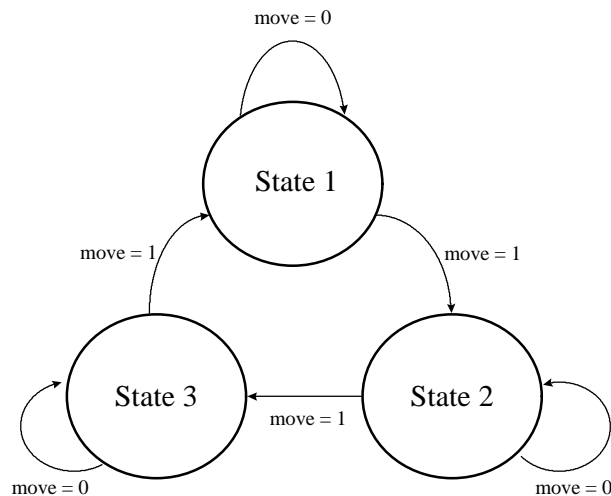


Figure 6: Example of a generic state machine with three states.

The 35VEE8 state machine has nine internal states each with its own output. The output of each state are control signals for a MUX that releases an address to some

location in the micro-store as well as a control signal for memory and I/O operations.

Once the current control word is asserted for a state, the state machine examines status signals from the data path and micro-store to determine the next state. At the next clock cycle, the state machine transitions to the next state.

The control unit has a special purpose register to hold the current address being accessed in the micro-store called a micro PC. The output of the micro PC goes to both the micro-store and an incrementer. The incrementer adds one to the current address and outputs to the MUX. This allows the state machine to release an incremental sequence of addresses to the micro-store. The inputs to the control unit MUX are starting addresses for the most commonly used micro-instructions. These include the fetch address (a fetch is a set of micro-instructions that retrieves the next instruction in memory and place it into the IR) and the micro-address of the current instruction. The control unit generates a starting micro-code address for the current instruction by taking the 8 bit instruction and mapping a new 16 bit micro-code address. A typical instruction might operate as follows. The fetch address is outputted from the MUX. The state machine allows the fetch address to be sent to the micro-store on the first clock cycle and then increments the address on each successive clock cycle until it detects the last fetch micro-instruction. It then releases the vectored address that was obtained from the IR incrementing it on successive clock cycles until detects the last micro-instruction. The state machine can then fetch a new instruction. There are some other details left out of this example like memory and I/O operations and resets and interrupts, but the state machine ultimately controls the operation of the computer.

Chapter 3. The Design Process

The design scenario for the 35VEE8 microprocessor is that of a small commercial firm competing for a government contract. A design specification provides the details of the microprocessor architecture as well as a parts list that can be used to implement the design onto hardware. Our group, consisting of three students, act as the design team for the project. Our goal is to present a finished design that implements a microprocessor with the correct functionality as well as to provide a good cost/performance ratio. The ratio is determined by the number of instructions executed per second divided by the cost of the design in dollars. Our team is free to implement the design in any way.

3.0 Interface

The processor has an external interface with 33 pins. There are eight bi-directional data lines, sixteen address output lines, six control lines, a clock input, and power and ground pins.

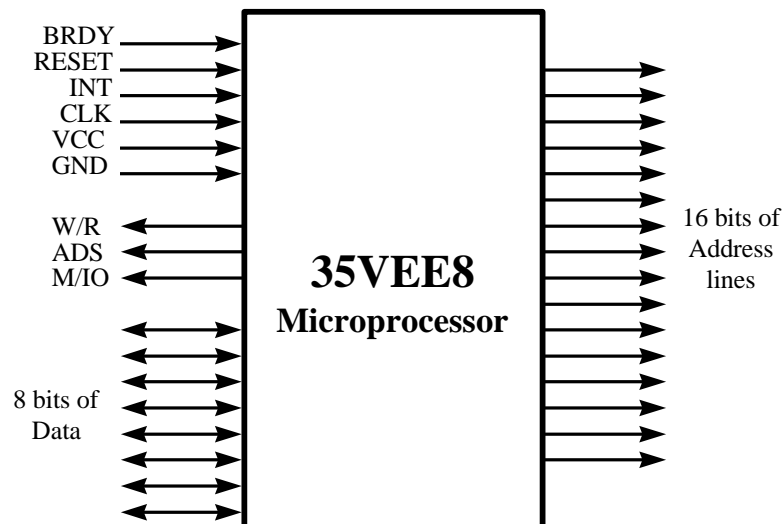


Figure 7: The 35VEE8 Microprocessor Interface.

The 35VEE8 is connected to the memory and I/O systems over common data, address and control buses. The memory and I/O systems each have their own controller to determine their participation in any read or write operation.

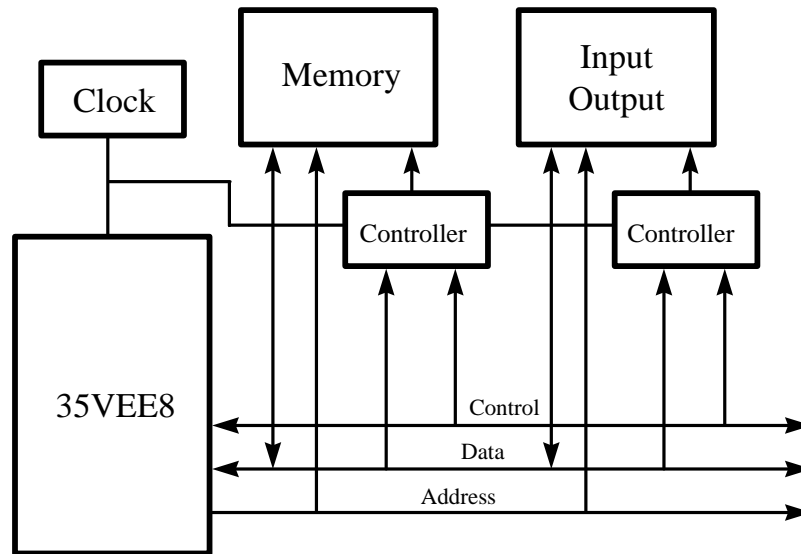


Figure 8: Diagram of 35VEE8 system.

3.1 Registers

A set of registers are available to the programmer of the 35VEE8 and need to be included in the design. Most instructions include the 8 bit accumulator register *A* which accepts all results generated by the ALU. Also required are a set of 8 bit general purpose registers, *B*, *C*, *D*, and *E*. The registers *D* and *E* also serve as a single 16 bit register, *X* where *D* holds the most significant bits and *E* holds the least significant bits. A 16 bit program counter, *PC*, is needed to keep track of the program execution and a sixteen bit stack pointer, *SP*, holds the address of a location in memory called the stack. The stack is a section of memory where data and retrieved on a first-in last-out basis during program execution. A 4 bit condition code register, *CC*, is required to hold current status

information generated by the ALU. The contents of the CC register are overflow, sign, carry, and zero in bit positions 3 down to 0.

3.2 Bus Protocols

Bus protocols dictate how the microprocessor communicates with the memory and I/O subsystems. The control signals for a read or write operation are generated by the microprocessor and are sampled on the rising edge of the clock. Logic at the memory and I/O subsystems can assert wait states until it finished with the current operation. When the external device is finished using the bus, it asserts a BRDY signal to the microprocessor. The penalty for violating bus protocols is the possibility of lost or incorrect data.

3.3 Reset and Interrupts

The reset and interrupt signals are generated by an external device and signal the microprocessor to execute some set of micro-instructions. When a reset signal is detected by the microprocessor, a new address is immediately loaded into the PC from memory locations 0 (least significant bits), and 1 (most significant bits) and interrupts are disabled. After a reset, the previous contents of the microprocessors registers can be in any random state. The reset is always executed at start up to set the microprocessor to some known state. It can be asserted at any other time to reset the microprocessor to that state.

An interrupt signal causes the microprocessor to complete the current instruction and then execute some set of instructions stored in memory. When an interrupt signal is detected and interrupts are enabled, a new address is loaded into the PC from memory locations 2 (least significant bits), and 3 (most significant bits).

3.4 Instruction Set

The design specification for the 35VEE8 provide for three different addressing modes and thirty-three separate instructions. Instructions require between one and three bytes (8 bits per byte) for both the instruction and data but the instruction itself is specified in the first byte. Instruction formats and descriptions are explained in detail in the *Programmers Reference Manual*.

3.5 Design Tools and Methodologies

Designing a microprocessor requires an efficient use of resources while following a detailed design methodology. This enables a design team to construct a processor in a timely manner while maximizing its efficiency using the processes of larger projects scaled down to the scope of the 35VEE8.

Mentor Graphics design tools are used to design and simulate all components at both a high and low level of abstraction. Low level abstractions are designs consisting of single logic elements that can be directly translated onto the hardware. Abstraction at the highest level uses a hardware description language (VHDL) to describe the behavior of components. These designs generate simulation results that can be used to test the validity of designs and also be used in system simulations with low level components when testing the overall design.

One of the design methodologies used throughout the design of the 35VEE8 is the Spiral Design process. This main aim of this process is to iterate through the design process multiple times in order to break the larger project into smaller more manageable tasks while developing risks and alternatives before every new iteration. The first iteration develops a high level of abstraction that tests the overall efficiency and validity of the

design. Each subsequent iteration breaks down the design down to into lower levels of abstraction. All work done in the EE435 portion of the project is considered one iteration of the spiral model. The problems and successes with that design provide a foundation for the next iteration which occurs in the EE436 portion of the project.

A methodology that works closely with the spiral model is the use of virtual prototypes. These working simulations test design decisions while allowing work to progress in the spiral model. Virtual prototypes are working models that can be comprised of different levels of abstraction. If one component is behind schedule for a certain system, a high level component can be constructed in order to simulate the entire system. This method can be used to speed the design process while testing a components interaction within a certain system.

Developing the microprocessor requires a dedicated group of people who establish and follow a design plan. This Integrated Product Development Team concept allows team members to work on their assigned tasks while combining resources to tackle more complicated issues. The strengths and weaknesses of tasks members are taken into consideration when assigning tasks while at the same time making sure all team members are sometimes challenged in areas that they are weak.

3.6 The First Spiral in EE435

The EE435 portion of the design project generated a complete virtual prototype developed at a high level of abstraction. The processor was almost completely described in VHDL using Mentor Graphics tools for both developing the VHDL code and simulating the results. Each student in the class worked individually to develop their

designs based on the *Programmers Reference Manual for the 35VEE8 Processor*. The manual contained all the information that needed to be included in the design.

Components for the processor were developed one at a time according to a class laboratory schedule. The first major component designed was the ALU. It included all required functionality with delays added into outputs to simulate that of logic components.

3.6.0 High Level Data Path

The next major component designed was the data path. The design was developed using only the list of needed registers and instruction set. A connectivity chart helped determine the relationship between components for data transfers. Before creating the chart however, each instruction was examined and a register transfer notation (RTN) description was created that described transfers of data from component to component.

The decomposition of an ADD instruction is shown in the following example:

ADD - Add the contents of register A to register (B,C,D,E) and then put the results in register A.

- 1) $ALU_x \leq Reg\ A$
- 2) $ALU_y \leq Reg(B,C,D,E)$
- 3) **ADD**
- 4) $RegA \leq ALU_z$
- 5) $CC \leq ALU_{carry_out}$

Example 1: RTN showing data transfer for an ADD instruction.

The connectivity chart was then constructed with needed components as both rows and columns. The rows indicated the source and the column indicated the destination. Each instruction and its RTN description was examined and a mark was placed in the corresponding row/column indicating a needed connection between components. Once finished, the connectivity chart showed the strengths (many marks in a row/column) and

weaknesses (few marks in a row/column) of needed connection paths between components. Since most components had many possible paths for receiving and sending their data, an efficient method was needed to allow transfers yet keep the number of control points low. For example, if a register had 8 possible components that could send it data, then a 8-to-1 MUX could be used to control whose its input it received. The problem with this example is that three control points are needed for the MUX. Multiply this number by all needed registers and the number of control points gets large. The design that was generated for the 35VEE8 instead used two major MUX's, one each for the 8 and 16 bit portions of the data path.

Some auxiliary components were also added to the design of the data path. A 16 bit incrementer/decrementer was added to allow adding/subtracting to both the PC and SP as well as a register to hold this result. Two 8 bit registers were also added to hold data temporarily until it can be transferred together to a 16 bit registers. Once the data path was constructed, Mentor Graphics simulations helped determine the correctness of the design.

3.6.1 High Level Control Unit

A control unit was then constructed to coordinate the movement and processing of data within the processor. A micro-programmed implementation was used to generate the control points because of its simplicity and ability to change. The components used include a state machine, instruction vector, microPC, incrementer, a MUX, registers, and a micro-store. The MUX controlled what address was released to the microPC who in turn released it to the micro-store. The MUX was controlled by the state machine who determined the correct address by examining the status bits from the data path,

micro-code bits, and the IR bits. The starting address for program instructions is accomplished by the instruction vector. It took as its input the data stored in the IR and generated an address for the micro-store. Once an address for any micro-instruction has been released, the incrementer adds one to the value held in the microPC to allow the sequential release of control points. A set of registers are hard-coded with starting addresses of certain micro-code instructions because these addresses are not generated by the IR. They output to the MUX for release by the state machine. To hold all the micro-instructions, set of connected ROM's release control points when presented with an address. The main testing of the control unit consisted of checking the state machine outputs for all possible inputs and then examining the release and sequence of control points. Since the control unit controlled all activities of the processor, its correctness was considered vital for a smooth integration with the data path.

3.6.2 High Level Virtual Prototype

All required functionality for the 35VEE8 was not required for the implementation of the EE435 design. A small program tested the design but proved that full functionality could be met with some modifications. This high level design proved sound and the long hours spent in its creation resulted in a familiarity with design tools as well as the design process.

3.7 EE436 Design Process

The result of the EE435 design was a functional design that simulated the behavior of the 35VEE8. The result of EE436 is a functional piece of hardware. Constraints that were absent in design considerations of EE435 are now rules that bind the design for EE436. This scenario is common in the processor design industry where space and speed

are major factors as well as time to market. The EE436 portion of the project involves teams of designers working with a limited number of hardware components trying to develop a fast and small design in a limited amount of time.

3.7.0 Design Team

Assigned teams of three students work together to realize the design of the 35VEE8. My design team is Greg Hughes and Diana Wong with Greg acting as team leader. Greg has the responsibility of meeting with Professor Aylor every Friday to discuss our team's progress. Our group follows the Integrated Development Team Process by which we work separately on tasks yet combine efforts when difficult areas arise. Additionally, all design decisions are reached by group consensus.

3.7.1 The ALU and Design Constraints with FPGA's

The first component designed at a lower level of abstraction is the arithmetic and logic unit. Our target hardware device for this device is an Actel FPGA with 547 modules and 69 pins.

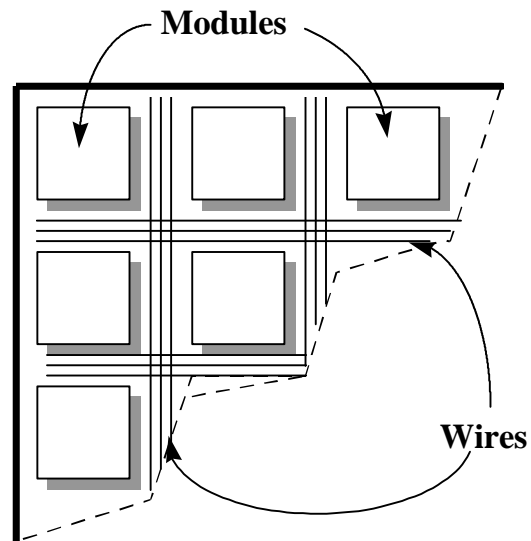


Figure 9: Example of an FPGA with Modules and Connecting Wires

Modules on the FPGA have the ability to act as almost any number of logic devices. For example, both a two input AND gate and a D Flip-Flop (storage device that holds one bit and is used in registers) each require just one module. Mentor Graphics has libraries of ACTEL parts of both hard (simple logic devices like AND gates) and soft (complex devices like multiple bit registers and adders) macros that can be used to build a device. Our design process has no concern over how the modules will be connected and laid out. The library parts are typical logic devices and simulate according to their Actel specifications. To implement a design onto an Actel FPGA, Mentor creates a file that is then used to physically and permanently fuse and route the design onto an FPGA with special hardware.

Our design group was provided with an Actel FPGA specification sheet that described each logic device and its module count. With a limited number of modules and pins for each FPGA, we need to optimize our design for both size and speed. The ALU is one of the more complex devices in the design and can be designed any a variety of ways. For example, the logic required to add 8 bits can be accomplished with 8 one bit full-adder devices with each having three inputs: the x and y to be added, and the carry-in. The output of each adder is a sum and a carry-out to the next adder.

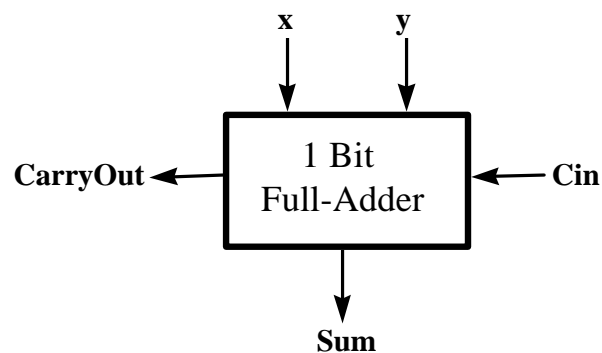


Figure 10: Example of a one bit adder.

The adders are connected so that each sum goes to the output and the carry-out goes to the next adder as a carry-in. This design has a low module count compared to a fast-adder type (more logic to for generate and propagate functions) but has a large delay associated with the propagation of the carry bits. Before generating the team design for our processor, we each designed individual ALU's as part of laboratory assignments. We combined the best aspects of each of our designs for the ALU. Included in our design were a eight full-adders whose outputs were also used to realize the OR and XOR functions needed for the ALU. The design was optimized as much as possible to achieve a good performance / cost ratio.

3.7.2 Data Path Design

Preliminary design of the data path occurred in conjunction of the ALU. Each of our team members compared our EE435 designs and discussed concerns over control points, external read/writes, partitioning, and simplicity of design. Our team set a goal of using only two FPGA's instead of the three that were provided. Partitioning the design meant that the data path would have to be divided onto separate FPGA's with the control unit added onto one of the two. We chose the design I had implemented as a basis for our data path. It was already somewhat divided into a eight bit and sixteen bit parts with MUX's and bus rippers (buses divided onto smaller sizes) that could be used transfer data between parts. Individual components like registers were assembled and from flip-flops with some registers having 2-to-1 MUX's added with no additional cost.

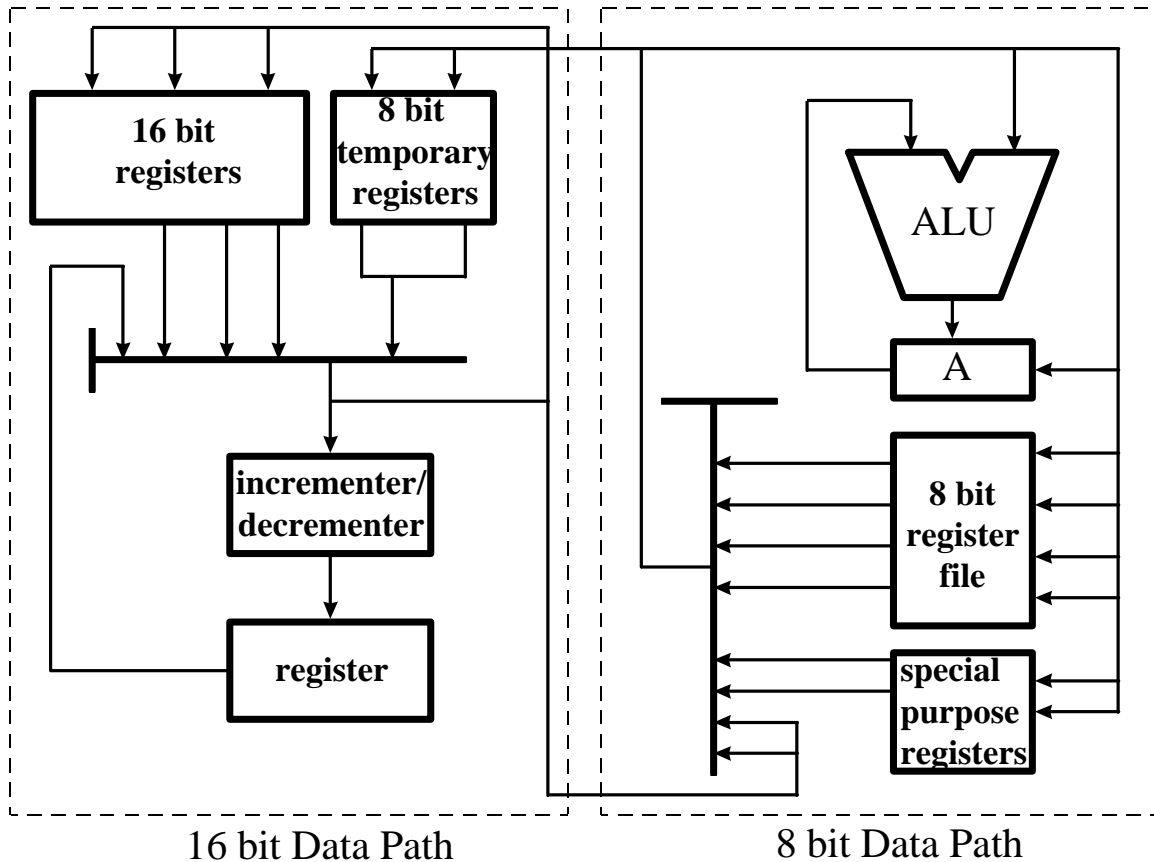


Figure 11: Partitioning of the Data Path.

To send and receive data from the MDR, the operations of a tri-state device (turns off the transfer of data on a bus) was added to allow bi-directional travel.

3.7.3 Control Unit Design

Design of both the data path and control unit occurred together. Work was begun on the assembly of the data path before the control unit but that result was constantly used to factor into the control unit design. Some preliminary factors discussed were the state machine, read/write operations, and control point generation and encoding. This helped our design team build the data path with knowledge of the interaction with the control unit. The design that was used for the control unit was similar to the design all of us had implemented in EE435. A MUX released a 16 bit address into a set of two ROMS that

acted as a micro-store. The ROMs are each 12 bits wide for a total of 24 bits for the micro-store. An incrementer adds one to the current micro-address which is fed back into the MUX. The MUX has six address inputs: the microPC +1, reset address, fetch address, PC+2 address (failed jump instruction), interrupt address and the instruction vector. Controlling the MUX is a state machine the has inputs from the data path and micro-store.

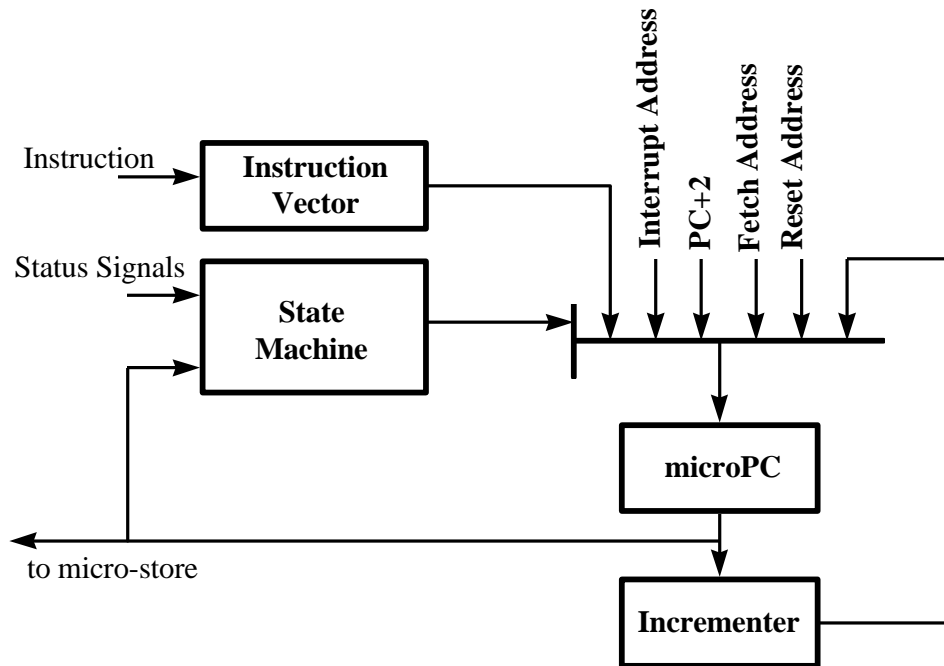


Figure 12: Diagram of the Control Unit

The state machine and instruction vector for the control unit were both written in VHDL. The code was then synthesized in Mentor Graphics to create logic for each device. The state machine required several prototypes to test its operation. The purpose of these prototypes was to initially to test its operation but eventually the state machine was integrated with the other components of the control unit to test overall operation.

The final state machine design has nine states as shown in figure 13.

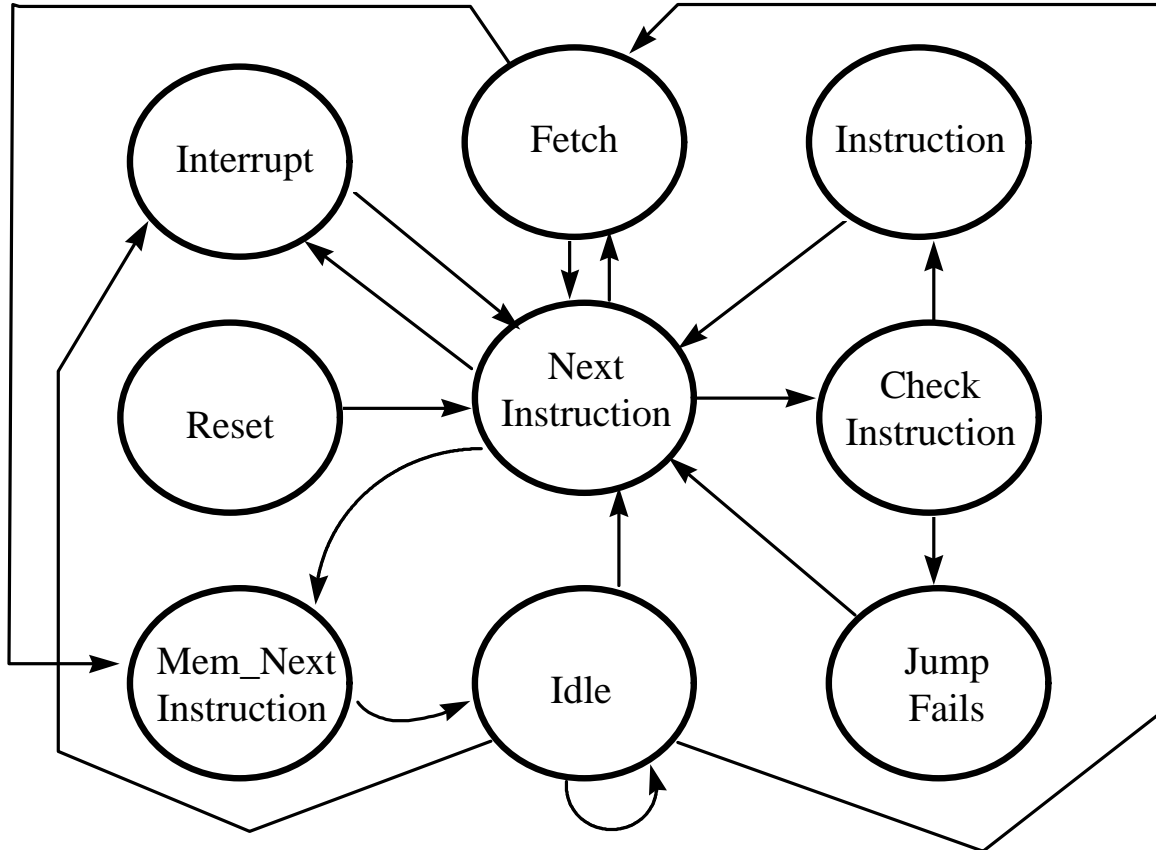


Figure 13: Simple Diagram of the State Machine for the 35VEE8.

Once the control unit was tested, the next hurdle to cross was the compacting of control points. The design has 33 control points total that needed to be encoded down to 24 bits.

All the control points are shown in below.

Registers:

- enable A
- enable B
- enable C
- enable D
- enable E
- enable CC
- enable MDR
- enable TempL
- enable TempH
- enable PC
- enable MAR
- enable SP

MUX's:

- select A
- select CC
- select MDR
- select Inc/Dec
- DataMUX(4)
- Inc/DecMUX(2)

Memory:

- W/R
- M/IO
- ADS
- tri-state

ALU:

- ALU_op(4)

OTHERS:

- end_bit
- fetch_bit

enable Inc/Dec Register

Example 2: Listing of all control points for the 35VEE8

To compact the control points, we examined all the instructions and looked for places where no two components were active on the same clock cycle. One control point that was eliminated is the tri-state device. When the state machine determines a read/write ($ADS = 0$) operation, it checks the W/R bit to determine the direction of the data lines and then generates the tri-state control signal. Some registers that are never enabled on the same cycle are *A*, *B*, *C*, *D*, *E*, *TempL*, and *TempH*. To choose one of these registers, a decoder device is used to select only one of several components.

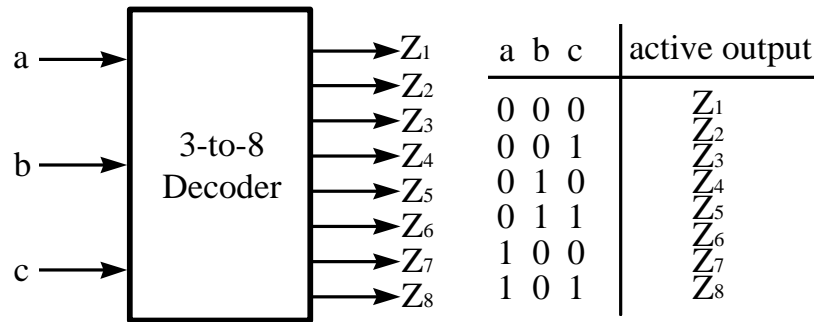


Figure 14: Example of a 3-to-8 Decoder.

Using an 8-to-1 decoder with the these seven registers compacts the seven control points down to three and allows the selection of one or none of the components (sometimes none of the registers need to be enabled). A 2-to-4 decoder was used to compact the three control points of the *PC*, *CC*, and *Inc/Dec* registers down to two. Sometimes certain components can use the same control points like the MUX's of *CC* and *A*. Combining their two control to one still allows for the separate enabling of the registers. The selection of an ALU operation and the Increment/Decrement components also never occur on the same cycle. Combining their six control points to only four still allows for the

separate operation of these components. The 24 compacted control points now fits within the storage capacity of the micro-store.

control point	output	controls
1	3-to-8 Decoder(0)	registers A, B, C,D ,E, TempL, TempH
2	3-to-8 Decoder(1)	
3	3-to-8 Decoder(2)	
4	A/CC MUX	MUX's of registers A and CC MDR register MUX of the MDR
5	enable MDR	
6	MDR MUX	
7	m(0) / IDMUX(0)	ALU operation and the 16 bit incrementer/decrementer
8	m(1) / IDMUX(1)	
9	m(2)	
10	m(3)	
11	DataMUX(0)	the MUX of the 8 bit data path
12	DataMUX(1)	
13	DataMUX(2)	
14	DataMUX(3)	
15	enable MAR	MAR register
16	2-to-4 Decoder(0)	registers PC, CC, Inc/Dec
17	2-to-4 Decoder(1)	
18	enable SP	SP register
19	select Inc/Dec	chooses increment or decrement
20	W/R	memory control signals
21	M/IO	
22	ADS	
23	fetch_bit	micro-store generated status signals
24	end_bit	

Figure 15: Compacted Control Points.

3.7.4 Integrated Data Path and Control Unit

After carefully assembling all the components of the data path and control unit, a count was made of the modules and pins. The 8 bit data path used approximately 52% of an FPGA at some 280 modules. The critical timing of the 8 bit data path requires 92ns (from the bi-directional port to through the 16-to-1 multiplexor into the ALU, and then latched into the A register). It was combined with the control unit onto a single FPGA with a total module count of approximately 480 (88.8% capacity) and a input/output pin count of 65.

The 16 bit data path counted some 260 modules at 48% of capacity and 39 input/output pins. The critical path timing for the 16 bit data path requires 112ns (from a 16 bit register through the 4-to-1 multiplexor, through the increment/decrementer and latched back into a register). All the components have been tested individually and testing of the integrated system is now planned.

Chapter 4. Improving Performance through Caching and Memory Access Control

There are several methods for processor designers to improve the memory access time of designs. Armed with the fact that smaller memories like registers are the fastest of all memories and external devices are the slowest, designers are able to arrange memory structures that can reduce the overall access time. A factor that designers must face are the smaller capacity and higher costs associated with fast memories. Another problem that designers can rectify is the higher access time for memory accesses in separate areas of memories.

4.0 Cache Memory

The principle behind cache memories is to provide a large virtual memory to the processor which in fact is some combination of memories with the fastest closest to the processor. The purpose of a cache is to hold data that the processor has used or generated most recently. Its success depends on the ability of the cache to reduce the average time required to retrieve data or instructions (Przybylski : 2).

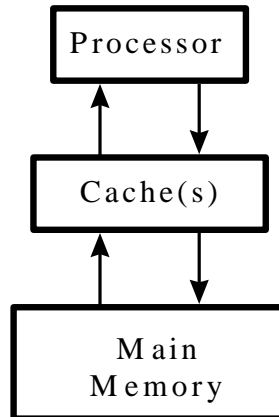


Figure 16: Diagram of the Cache Memory Hierarchy.

Temporal locality is the expectation that data accessed once will be accessed again in the near future. A control device exploits the pattern of memory use so that once the processor makes a memory access, a chunk of data is placed into a cache. Spatial locality is the expectation that data located together in memory will be accessed within a small amount of time of each other. Typically programs exhibit both high temporal and spatial locality. Caches exploit this fact and maintain data of both the temporal and spatial variety. A cache miss refers to the absence of requested data within the cache while a cache hit refers to its presence. The number of cache misses divided by the total number of memory data requests is called the miss ratio. The consequences of a cache miss often means that the processor sits idle waiting for data. When a program has a large miss ratio, the processor sits idle for a large segment of the program and the program takes longer to execute. Because of this, the miss ratio is the equation that is used to describe the effectiveness of a cache design.

Because caches are smaller than main memory, they cannot hold all the possible data that a program will use. To allow the swapping of data blocks from main memory and the cache, there are several replacement schemes. These come into play when data recently accessed maps into a filled cache slot. The least-recently used (LRU) scheme is a simple method that kicks out a block that hasn't been accessed for the longest amount of time. Another simple scheme is first-in first-out (FIFO). It replaces the block that has been in the cache the longest. A more complex scheme is the least-frequently used (LFU) which uses a counter to replace the block with the lowest number of references.

There are a variety of methods for designing the size and organization of caches. The optimal design has a cost per bit that is close to that of the main memory and an

average access time close to that of the cache access time (Stallings: 157). Three different methods are available for the mapping blocks of main memory data into the cache. A direct mapped cache allows each block of main memory only one slot in the cache. It is fairly simple and requires less hardware to implement. The main disadvantage of direct mapped caches is that if two main memory blocks used by a program map into the same block, then the cache will constantly swap the two blocks.

A mapping scheme that solves the swapping problem of the direct mapped cache is the associative mapped cache. It allows a main memory block to be mapped into any slot in the cache. Replacement schemes come into play for this method and are used to minimize the miss ratio. The disadvantage of this method is that it requires more hardware to examine each slot in the cache.

The set associative cache combines the positive attributes of both the associative cache and the direct mapped cache. The cache is divided into sets with each set having a number of slots. This method decrease the amount of hardware required to examine all the slots since a block can only be within one set.

The cache size is one of the most important factors that effect its performance. For the direct mapped cache, its performance increases as the size of the cache increases up to a certain size where eventually it levels off (Przybylski : 46). Associative mapped caches have a better overall performance compared to direct mapped caches except for small sized caches. The best overall performance is achieved by the set associative cache who again only suffer in comparison to direct mapped caches of small sizes (Przybylski : 60).

4.1 Memory Access Control

One problem with caches is that they do not attempt to improve the performance of the memories themselves. Traditionally RAM's received their name from the fact that random accesses of memory took the same amount of time. This is no longer the case with most current DRAM memories (Dynamic Random Access Memory) which use a page-mode of operation. The behavior of a page resembles that of single cache line so that an access outside of the current page causes a new page access. This in turn creates an overhead of setting up and servicing the new page (McKee(7): 3). Armed with the fact that modern memories no longer display uniform access times, designers can exploit these characteristics to optimize performance (McGee: 1).

An area where caching schemes do not produce good performance is for vector like applications (McKee(7): 1). The access of vector computations are predictable but suffer in cache implementations because they suffer poor temporal locality (McKee(8): 1). Vector computations are characterized by a linear transversal of vector-like data where a single reference to each element of the vector occurs only once during long parts of the computation. These streams occur during execution "loops", a repeated execution of a sequential instructions. The complete reference pattern of these vectors can be determined during program execution just before the loop is executed. Vectors are not found just in scientific calculations, their use is common in such applications as graphics applications, database queries, string processing, and many multimedia applications (McKee(7): 2). This fact makes vector processing more applicable to modern computing applications.

A technique that could be used to optimize the performance of both regular and stream memory accesses involves separating the data into their own memories. Streamed

data has been shown to exhibit poor cache behavior so it is stored in a separate buffer. Both the problem of disparate memory access times and vector memory accesses can be addressed with some additional hardware and software. Figure 16 shows a diagram the memory access control system developed at *The University of Virginia* by members of both the Department of Computer Science and Electrical Engineering. It is characterized by a memory scheduler that reorders the memory accesses to maximize the number of bits per second, or bandwidth. When a program is about to enter a loop where stream data is accessed, the compiler provides the scheduler with information that helps it maximize bandwidth while filling the buffer unit. The cache is responsible for non-stream data so does not suffer from the poor performance of stream data. The name given to this system is the Stream Memory Controller.

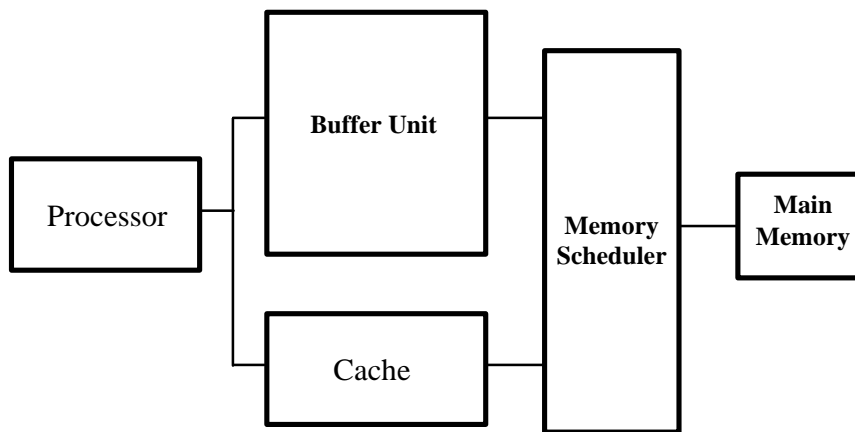


Figure 17: Diagram of Stream Controller System (McKee(8): 2)

The Buffer Unit includes high speed buffers to store stream data and registers to provide the Memory Scheduler with stream parameters. The speed of the Buffer Unit is not part of the critical path of memory accesses (McKee(7): 7). One method of buffering the stream data within the Buffer Unit is FIFO's. Under this implementation the FIFO's act as personal buffers for each stream of data (McKee(7):8). After all the scheduling and

buffering, the processor receives the data from the memory in the natural order that it was requested. The software for this implementation includes a simple addition to the compiler so that it can detect streams.

The evaluation of the memory controller at UVA included testing the system with several benchmark programs using differing FIFO depths, more than one memory, and varying DRAM speeds. Compared to non-memory controlled systems, the memory controller exhibited large improvements in bandwidth usage while approaching 100% at FIFO sizes of 256 units (McKee(7) :11). All this was accomplished with some known compiler technology and little additional hardware.

Chapter 5. Conclusions

5.0 Summary

The entire 35VEE8 project provides students the opportunity to both participate and lead in the design of a processor. Producing a working processor was the main focus of our efforts but important concepts such as design methodologies, design tools, digital design, and problem solving were factored into the design process. The simple 8 bit microprocessor includes a data path and control unit can interfaces with both memory and I/O. The data path stores and processes all the data within the processor and the control unit coordinates the movement and processing of the data and the interfacing with the memory and I/O units. A memory controller coordinates the operation of both the RAM and ROM memories provide and receive data and instructions for the processor. An I/O controller coordinates the actions of the UART and driver to format data exchanges between the processor and the external world. There are 3 different addressing modes for the 33 instructions.

Individually I researched memory problems that can be solved by both caches and memory controllers. Caches showed an increase in performance, especially large sized set-associative caches. Memory controllers for vector like stream data used along with caches show in improvement performance across the board.

5.1 Interpretation

Cache memories are not meant to be used for all types of memory accesses. They are limited by the scheme that controls the storage and size of the data blocks. When used with stream data they exhibit poor performance. Stream data is not the only type of data

that causes poor performance with processors. The fact that many access streams can not be predicted causes the miss ratio to be what it is. Memory control used in conjunction with caches removes the poor behavior of streams from the cache. The cache within the stream memory control system now only has to deal with the remaining unpredictable behavior of program execution. This not only means an increase in speed for scientific calculations, but an increase in other more common applications like multi-media who use vectors in their programs. My analysis does not intend to conclude that the memory controller for streams can solve the memory bottleneck, but it does provide a method for improving cache performance with low cost.

5.2 Recommendations

My project examined the problems with cache methods and one way in which a memory controller can remove seemingly unpredictable behavior. This opens up a new area for examination: poor cache behavior not involving streams of data. Until memory speeds match that of the processor, methods have to be devised to feed a constant supply of data to the processor.

Chapter 6. Bibliography

1. Ashendon, Peter J. The Designers Guide to VHDL. Morgan Kaufmann Publishers, Inc. San Francisco, California. 1996.

Programming manual for the VHDL language.

2. Aylor, James. Class notes - EE436. University of Virginia, 1997.
3. Hamacher, V. Carl and Zvonko G. Vranesic and Safwat G. Zaky. Computer Organization. 3rd Edition. McGraw-Hill, Inc. New York, NY. 1990

Textbook that describes computer components and their operation including memories, micro-code, assembly language, and logic operations.

4. Hennessy, John L. and David A Patterson. Computer Architecture: A Quantitative Approach. Morgan Kaufmann Publishers, Inc. San Francisco California. 1990.
5. Herzog, James H. Design and Organization of Computing Structures. Franklin Beedle And Associates, Inc. Wilsonville, Oregon. 1996.

Textbook that describes the operation and design of digital computer components. Included are descriptions of the data path and control unit.

6. S.W. McGee, R.H. Klenke, J.H. Aylor, and A.J. Schwab. Design of a Processor Bus Interface ASIC for the Stream Memory Controller Proc. IEEE International ASIC Conference (ASIC'94), Rochester, NY, September 1994.
7. S.A. McKee, R.H. Klenke, A.J. Schwab, Wm.A. Wulf, S.A. Moyer, C. Hitchcock, and J.H. Aylor. Experimental Implementation of Dynamic Access Ordering. Proc. IEEE 27th Hawaii International Conference on Systems Sciences (HICSS-27), Maui, HI, January 1994.
8. S.A. McKee, A. Aluwihare, B.H. Clark, R.H. Klenke, T.C. Landon, C.W. Oliver, M.H. Salinas, A.E. Szymkowiak, K.L. Wright, Wm.A. Wulf, and J.H. Aylor. Smart Memory = Better Performance: Improving Effective Bandwidth for Streams. Submitted for publication, February 1996.

An article describing the Smart Memory Controller, outline of project, and highlights of results.

9. Roth, Charles H. Fundamentals of Logic Design 4th Edition. West Publishing Co. St. Paul, Minnesota. 1992

Textbook describing the most basic of logic components like flip-flop, MUX's, logic gates, and decoders. Also of importance are timing issues and the design and operation of state machines.

10. Stallings, William. Computer Organization and Design. Macmillan Publishing Co. New York, NY. 1990.

Textbook that involves more of the architecture side of computers. Includes information on memories and caches.

11. Przybylski, Steven A. Cache Memory Hierarchy Design: A Performance Directed Approach. Morgan Kaufmann Publishers, Inc. San Mateo, California. 1990.

An in depth description of caches and characteristics that effect their performance.

12. Williams, Ronald D. Programmer's Reference Manual for the 35VEE8 Processor. Version 1.6. 1995.

The specification sheet for the design of the 35VEE8. Included are diagrams of the processor interface, bus protocols, system diagram, instruction set, and instruction format.

13. "EE436 Homepage" (<http://csis.ee.virginia.edu/~rhk2j/ee436/materials.html>)