International Journal of Electronics, Communication & Soft Computing Science and Engineering
ISSN: 2277-9477, Volume 3, Issue 4

Review of 5 stage Pipelined Architecture of 8 Bit Pico Processor

Shankar Kumar Mishra Dr. Nisha P Sarwade

Abstract- **Proposed paper is the study of unpipelined architecture of a 8 bit Pico Processor (pP) [3][4] and how its overall through put can be increased by implementing pipelining. Pico processor is an 8 bit processor which is similar to 8 bit microprocessors for small embedded applications and it is intended for educational purpose .In the past un pipelined single cycle and multi cycle Pico Processor is implemented [3] .Its speed and overall through put can be increased by implementation of pipeline architecture [1] so that it can be used in small embedded applications like gaming processor.**

Keywords- **Pico Processor, VHDL, RISC, Pipeline.**

I. INTRODUCTION

The picoProcessor abbreviated as pP is an 8-bit processor intended for educational purposes. It is similar to 8-bit microprocessor for small embedded applications, but has an instruction set architecture more similar to RISC processors. The pP has separate instruction and data memories. The instruction is 4K instruction in size and the data memory is 256 bytes, the pP can also address I/O devices using up to 256 input and output ports. Within the processor there are eight 8-bit general purposes registers r0 to r7. Register r0 is always read as zero and ignores writes. In the past first unpipelined single cycle architecture of pP is implemented then it has been observed that its speed can be increased by introducing multi cycle unpipelined architecture [3].

Apart from past work it is observed that modern day processors are very fast in order to be able to process a large number of instructions. One way of making the processor fast is to increase the clock frequency. However, since the power dissipation of a microprocessor is proportional to the frequency of the clock, having a very high clock could lead to overheating of the processor. Another way to increase the number of instructions is to process multiple instructions at the same time. Pipelining is an implementation technique whereby multiple instructions are overlapped in execution. A pipelined processor increases performance by increasing the throughput as compared to a non-pipelines processor. Pipelining, however, introduces a whole new set of problems, also referred to as hazards, while executing the instructions. The following paper describes the study of un pipelined single cycle and multi cycle architecture of picoProcessor and how its overall through put can be increased by introducing pipelinig[1][2].In this paper section I consists of introduction to picoProcessor and pipelining, Section II describes the un pipelined single cycle picoProcessor section III tells the multi cycle unpipelined architecture, section IV describes the proposed architecture with pipeline, section V gives the hazards in pipeline architecture of pP whereas section VI gives the expected result and future work.

II. AN UN-PIPELINED SINGLE CYCLE pP

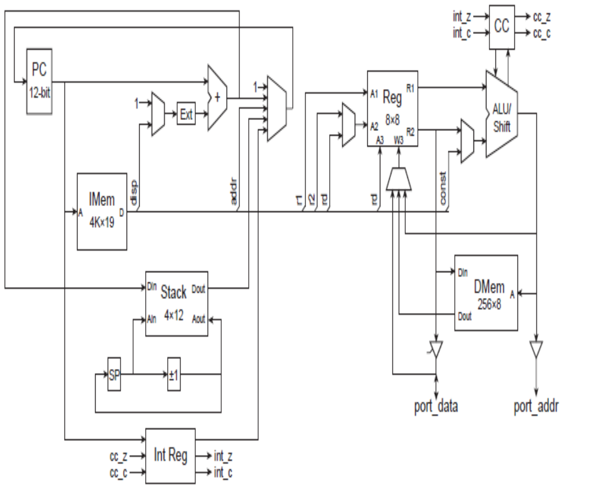

FIGURE 1 Unpipelined, single-cycle organization.

Figure 1 shows an unpipelined pP organization that executes each instruction in one clock cycle. Values stored on one clock edge flow through the data path, and the machine state is updated on the next clock edge. The clock Period must be long enough for the slowest path through the design. Execution within a cycle starts with checking whether an interrupt request is pending. If one is, the current PC and condition code bits are saved in the interrupt register and the next PC value is selected to be the address 1. No other machine state is updated. If no interrupt is requested, the PC value is used to index the instruction memory to fetch the instruction to be executed. Since all operations for the instruction take place within a cycle, the instruction memory must be an asynchronous ROM. The next PC value depends on the instruction op code and, in the case of branch instructions, whether the branch is taken or not. For JSB instructions, the next PC value is saved into the return-address stack and the stack pointer is incremented. For RET instructions, the top value in the stack is used as the next PC and the stack pointer is decremented. The general-purpose register (GPR) is an a multiport register file with two asynchronous read ports and a synchronous write port. The register address field for one read port is the r1 field of the instruction. The address for the other read port is either the rd. field (for STM and OUT instructions) or the r2 field (for other instructions).The write port is used for ALU, shift, load and input instructions, provided the destination address is not r0. The rd field from the instruction is used as the write-port address, and the data to be written comes from the appropriate source, depending on the instruction [3].

The ALU calculates the result value for ALU and shift instructions and the effective address for memory instructions. For ALU and shift instructions, the condition code bits are updated according to the result. The multiplexer on the ALU input selects between a register operand for register-register instructions or the constant value from the instruction for immediate and memory instructions. The data memory is asynchronously read for load instructions and synchronously written for store instructions. The address comes from the ALU result. The external port interface is used for input and output instructions. Since this implementation of the pP executes instructions within a single cycle, it assumes that port inputs are asynchronous within a cycle and that port outputs update the port register synchronously at the end of the cycle. Thus, the port ready input is ignored.

III. AN UNPIPELINED MULTICYCLE pP

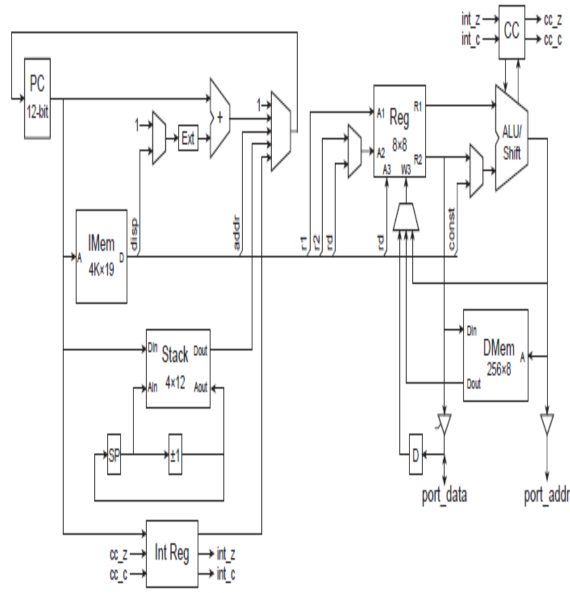

FIGURE 2 An unpipelined multi-cycle organization for the pP.

Figure 2 shows an unpipelined pP organization that takes multiple clock cycles to execute each instruction. On each cycle, one step of instruction interpretation is performed, and the machine state is updated at the end of the cycle. Different instructions may take different numbers of cycles, depending on the interpretation steps required. The advantage of this approach over the single-cycle approach is that much less work needs to be done per cycle, so the cycle time can be faster. Furthermore, many instructions do not require all interpretation steps, so their execution will be faster than for the single-cycle implementation [3].

The first cycle of execution involves checking whether an interrupt request is pending. If one is, the current PC and condition code bits are saved in the interrupt register and the PC is set to 1. Execution of the interrupt service code then proceeds in the subsequent cycle .If no interrupt is requested, the first cycle is used to index the instruction memory to fetch the instruction to be executed. In this implementation, the instruction memory is a synchronous ROM, and the ROM output register forms the instruction register (IR). The PC register is updated with the incremented PC value. During the second cycle, the GPR register file is accessed to fetch operands, in case they are required. The register file in this implementation has synchronous read ports, and the operands are stored in two output registers. Also in this cycle, control-flow processing is performed. If the instruction in the IR is a conditional branch that is taken, the PC is updated with the sum of its current value and the branch displacement. If the instruction is a JMP, the PC is updated with the target address. If the instruction is a JSB, the PC is updated with the target address, the current PC value is pushed onto the return-address stack, and the stack pointer is incremented. If the instruction is a RET, the PC is updated from the top of the stack, and the stack pointer is decremented. If the instruction is a RETI, the PC and condition codes are restored from the interrupt register, and interrupts are enabled. If the instruction is an ENAI or DISI, the interrupt enable bit is set accordingly. In all cases of control flow instructions, processing is complete after the second cycle.

The third cycle (if required) involves computation of a data result or an effective address by the ALU. The result is stored in an output register. Also, for arithmetic, logic and shift instructions, the condition code bits are updated. For memory and I/O instructions, a further cycle is used to access the memory or port register. The ALU output register is used as the address. The data memory in this implementation reads and writes synchronously. For memory stores, write data from the GPR register file output register is stored at the end of the clock cycle. For memory loads, read data is made available at the data memory output register at the end of the cycle. For port input and output instructions, the pP checks the port ready input at the end of the cycle. If it is negated, the pP repeats the cycle, allowing the port controller extra time to read or write the data. When port ready is active, the input or output operation is complete. For input instructions, the port data is stored in the data input register. A final cycle is required for instructions that update a destination register in the GPR register file, namely, arithmetic, logic, shift, and load and input instructions. The data source is one of the ALU output, the data memory output or the port input data register, depending on the instruction. The destination register (if not r0) is updated at the end of the cycle.

IV. Proposed Architecture With Pipeline

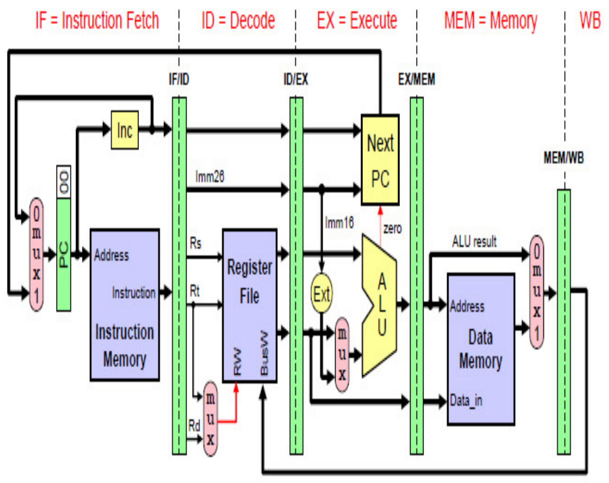

Fig.3 pipeline architecture of 8 bit Pico Processor

4.1 Instruction Fetch Unit

The first stage in the pipeline is the Instruction Fetch. Instructions are fetched from the memory and the Instruction Pointer (IP) is updated. The function of the instruction fetch unit is to obtain an instruction from the instruction memory using the current value of the PC and increment the PC value for the next instruction as shown in Figure. This stage is where a program counter will pull the next instruction from the correct location in program memory. In addition the program counter will updated with either the next instruction location sequentially, or the instruction location as determined by a branch The instruction fetch stage is also responsible for reading the instruction memory and sending the current instruction to the next stage in the pipeline, or a stall if a branch has been detected in order to avoid incorrect execution. The instruction fetch unit contains the following logic elements that are implemented in VHDL: 12-bit program counter (PC) register, an adder to increment the PC by four, the instruction memory, a multiplexor, and an AND gate used to select the value of the next PC. Program counter and instruction memory are the two important blocks of Instructions Fetch Unit.[1][2]

4.1.1 Program counters (PC): It is a 12 bit device that is connected to the data bus and the address bus. It will hold its value unless told to do something. If the I/P is kept high the device will count.

4.1.2 Instruction memory (IM): The Instruction memory of pP is of 4KB. During the Instruction Fetch stage, a 19-bit instruction is fetched from the memory. The PC predictor sends the Program Counter (PC) to the Instruction memory to read the current Instruction. At the same time, the PC predictor predicts the address of the next instruction by incrementing the PC by 1.

4.1.3 Instruction registers (IR): An instruction register (IR) is the part of control unit that stores the instruction currently being executed or decoded. In simple processors each instruction to be executed is loaded into the instruction register which holds it while it is decoded, prepared and ultimately executed, which can take several steps. Traditional RISC processors use a pipeline of instruction registers where each stage of the pipeline does part. Of the decoding, preparation or execution and then passes it to the next stage for its step. Modern processors can even do some of the steps of out of order as decoding on several instructions is done in parallel. Decoding the op-code in the instruction register includes determining the instruction, where its operands are in memory, retrieving the operands from memory, allocating processor resources to execute the command. The output of IR is available to control circuits which generate the timing signals that controls the various processing elements involved in executing the instruction.

4.2 Instruction Decode Unit

The Instruction Decode stage is the second stage in the pipeline. Branch targets will be calculated here and the Register File, the dual-port memory containing the register values, resides in this stage. The forwarding units, solving the data hazards in the pipeline, reside here. Their function is to detect if the register to be fetched in this stage is written to in a later stage. In that case the data is forward to this stage and the data hazard is solved. This stage is where the control unit determines what values the control lines must be set to depending on the instruction. In addition, hazard detection is implemented in this stage, and all necessary values are fetched from the register banks. The Decode Stage is the stage of the CPU's pipeline where the fetched instruction is decoded, and values are fetched from the register bank. It is responsible for mapping the different sections of the instruction into their proper representations (based on R or I type instructions). The Decode stage consists of the Control unit, the Hazard Detection Unit, the Sign Extender, and the Register bank, and is responsible for connecting all of these components together. It splits the instruction into its various parts and feeds them to the corresponding components. Registers Rs and Rt are fed to the register bank, the immediate section is fed to the sign extender, and the ALU op-code and function codes are sent to the control unit. The outputs of these corresponding components are then clocked and stored for the next stage The Control unit takes the given op-code, as well as the function code from the instruction, and translates it to the individual instruction control lines needed by the three remaining stages. This is accomplished via a large case statement.

4.2.1 Control unit: The control unit of the Pico processor examines the instruction op code bits [19 – 14] and decodes the instruction to generate control signals to be used in the additional modules. The RegDst control signal determines which register is written to the register file. The Jump control signal selects the jump address to be sent to the PC. The Branch control signal is used to select the branch address to be sent to the PC. The MemRead control signal is asserted during a load instruction when the data memory is read to load a register with its memory contents. The MemtoReg control signal determines if the ALU result or the data memory output is written to the register file. The ALUOp control signals determine the function the ALU performs. (E.g. and, or, add, shl, shr) The MemWrite control signal is asserted when during a store instruction when a registers value is stored in the data memory. The ALUSrc control signal determines if the ALU second operand comes from the register file or the sign extend. The RegWrite control signal is asserted when the register file needs to be written.

4.2.2 Register files (RF): During the decode stage, the two register Rs & Rt are identified within the instruction, and the two registers are read from the register file. In this design, the register file is of 8X8 size (namely r0 to r7) which had 8 entries. At the same time the register file was read, instruction issue logic in this stage determined if the pipeline was ready to execute the instruction in this stage. If not, the issue logic would cause both the Instruction Fetch stage and the Decode stage to stall. If the instruction decoded was a branch or jump, the target address of the branch or jump was computed in parallel with reading the register file. The branch condition is computed after the register file is read, and if the branch is taken or if the instruction is a jump; the PC predictor in the first stage is assigned the branch target, rather than the incremented PC that has been computed.

4.3 Execution Unit: The third stage in the pipeline is where the arithmetic- and logic-instructions will be executed. All instructions are executed with 8-bit operands and the result is a 8-bit word.The execution unit of the pP processor contains the arithmetic logic unit (ALU) which performs the operation determined by the ALUop signal. The branch address is calculated by adding the PC+1 to the sign extended immediate field shifted left 2 bits by a separate adder. The logic elements to be implemented in VHDL.

4.3.1 ALU unit: The arithmetic/logic unit (ALU) executes all arithmetic and logical operations. The arithmetic/logic unit can perform four kinds of arithmetic operations, or mathematical calculations: addition, subtraction, multiplication, and division. As its name implies, the arithmetic/logic unit also performs logical operations. A logical operation is usually a comparison. The unit can compare numbers, letters, or special characters. The computer can then take action based on the result of the comparison. This is a very important capability.

4.4Memory Access unit

The memory access stage is the fourth stage of pipeline. This is where load and store instructions will access data memory. During this stage, single cycle latency instructions simply have their results forwarded to the next stage. This forwarding ensures that both single and two cycle instructions always write their results in the same stage of the pipeline, so that just one write port to the register file can be used, and it is always Available. If the instruction is a load, the data is read from the data memory.

4.4.1 Data Memory Unit (DM): The data memory unit is only accessed by the load and store instructions. The load instruction asserts the MemRead signal and uses the ALU Result value as an address to index the data memory. The read output data is then subsequently written into the register file. A store instruction asserts the MemWrite signal and writes the data value previously read from a register into the computed memory address.

4.5 Write back unit

During this stage, both single cycle and multi cycle instructions write their results into the register file.

V. Various Hazards In Pipeline

Although with the help of pipeline we can increase the clock rate and overall throughput of a processor but during the process the following hazards may occur [1]

5.1 Structural hazards: - It arises from resources conflict when the hardware cannot support all possible combinations of instructions simultaneously in overlapped execution. It can be handled by using separate memories one for data and one for instructions.

5.2 Data Hazards: - When an instruction depends on the results of a previous instruction in a way that is exposed by the overlapping of instructions in the pipeline. It can be handled by using data forwarding.

5.3Control Hazards:- It arises from the pipelining of branches and other instruction that change the PC.

VI. Conclusion And Future Work

With the help of pipeline architecture mentioned above overall throughput of the pP can be increased.As far as result is concerned a basic 8 bit microprocessor having clock rate of 2 MHZ and consists 6000 transistors achieve the MIPS of 0.64(INTEL 8080).So upon pipelining the Pico processor should achieve greater MIPS than above. With this kind of improve performance pP can be used for small embedded applications as well as for educational purpose.

Future work will be added by increasing the number of instructions and to add more pipelined stages in the design to improve the performance of the design and to increase the speed of the processor.

References

[1] John L. Hennessy, David A. Patterson "Computer Architecture Quantitative approach"4th edition Morgan Kaufmann publishers.

[2] John L. Hennessy, David A. Patterson "Computer Organization and Design The hardware Software Interface"3rd edition Morgan Kaufmann publishers.

[3] Peter J. Ashenden. "Pico Processor model" Elsevier book store http://booksite.elsevier.com/9780124077263/picoprocessor.php

[4] Ken chapman "Pico Processor" Ken Chapman IEE Computing and Control Engineering October /November 2003 .

[5] Galani Tina, R.D.Daruwala "Performance Improvement of MIPS Architecture by Adding New Features" IJARCSSE,vol 3,Issue 2,feb 2013.

[6] Kirat Pal Singh,Shivani Parmar "Vhdl Implementation of a MIPS 32 bit pipeline processor" IJAER, vol. 7,No.11(2012).

[7] R.Uma "Design and performance Nalysis of 8-bit RISC Processor Using XILLINX Tool" IJERA,Vol.2 Issue 2,Mar-Apr 2012,pp.053-058.