**Overview**

In this project we have developed an Accumulator Based Processor. This processor “Hybrid Architecture” because it is the “Accumulator (ACC) based type ISA” and it is also implemented as a two stage pipelined architecture. The first stage was used to fetch the instructions from the Instruction RAM. The second stage was used to execute the instructions. Our processor used separate Instruction RAM and Data RAM to support the hybrid pipelined ACC Architecture. To get the additional project points, we also developed all of the other enhancing features such as Cache, MHVPI, assembler, two-way handshake, etc.

We worked hard to have as many performance enhancing feature in out processor as we could. And we were able to have all of those feature implemented. As a result, we were able to meet the minimum required features, and also have all the additional enhancing features implemented in our processor. We have also exhaustively tested all of our instruction with all of our addressing mode and we can ensure you that our processor is working correctly.

**Achieved, Non-Achieved Goals**

When we began this project, we never thought that we would be able to achieve all of our goal. We were just trying to do our best. Surprisingly, we were able to finish all of the required features, and all of our goals. One of the things that we thought we would never be able to implement the cache. However, after spending so much time we were able to create a Direct Mapped Cache (with write through policy.) We were also able to implement other additional features as well. We even make an “Assembler” for our ISA, and we didn’t have that as our goal in the first place. Therefore, we were able to achieve all of the required functions; furthermore, we were able to achieve all of additional functions to enhance the performance of our processor.

The additional enhancing features we achieved are:

* Two Stage Pipelined Architecture
* 3 Addressing Modes: Direct, Indirect, and Immediate addressing mode
* Direct Mapped Cache (Write Through Policy)
* Separate Instruction and Data Memories Supporting Hybrid Pipelined Architecture
* Assembler
* MHVPI system

Therefore, we don’t have any non-achieved goals. Because we completed everything.

**INSTRUCTION FORMAT**

|  |  |  |
| --- | --- | --- |
| Opcode (5 bit) | Addressing Mode (3) | Oprand (8) |

**Addressing Mode**

|  |  |  |  |
| --- | --- | --- | --- |
| Addressing mode | Machine Code | Example instruction | Meaning |
| Immediate | 000 | ADD value | ACC <= ACC+ X |
| Direct | 001 | ADD @value | ACC <= ACC+ MEM[X] |
| Indirect | 010 | ADD #value | ACC <= ACC+ MEM[MEM[X]] |

**Assembly Codes and Documentations**

|  |  |  |  |
| --- | --- | --- | --- |
| OP Code | Assembly Language Instruction | RTL Description | Documentation |
| 00000 | ADD X | ACC=ACC + X | Perform the ALU operation as described in the column to the left between the value in the Accumulator and the direct/indirect memory address value or the immediate value. |
| 00001 | SUB X | ACC=ACC-X |
| 00010 | OR X | ACC=ACC or X |
| 00011 | ORNOT X | ACC=ACC or ~X |
| 00100 | AND X | ACC=ACC or X |
| 00101 | ANDNOT X | ACC=ACC and ~X |
| 00110 | COMPACC | ACC= ~ACC | Complement the value in the Accumulator, then store the value back into the Accumulator. |
| 00111 | COMP X | ACC= ~X | Complement direct/indirect memory address value or the immediate value, then store the value back into the Accumulator. |
| 01000 | LOAD X | ACC= X | Load the Oprand into the Accumulator. |
| 01001 | STR X | MEM[X] = ACC | Store the value in the Accumulator to the Memory location X. |
| 01010 | SHIFTL | ACC= ACC<<1 | Left shift the value in the Accumulator. |
| 01011 | SHIFTR | ACC= ACC>>1 | Right shift the value in the Accumulator. |
| 01100 | INPUT | ACC = Bus | Load the value from the bus into the Accumulator with two-way hand shaking protocol utilized. |
| 01101 | OUTPUT | Bus = ACC | Output the value in the Accumulator to the Bus with two-way hand shaking protocol utilized. |
| 01111 | MASKHVPI X | MASKHVPI = X | Mask the value in the HVPI Register with the value X.  When the HVPI Reg has the value:  0001 overflow interrupt has the highest priority.  0010 Bad Instruction interrupt has the highest priority.  0100 Bad Addressing Mode interrupt has the highest priority.  1000 External interrupt has the highest priority.  If the HVPI Reg is not set,  the priority will be given in this order:  Overflow,  Bad Insturction,  Bad Addressing Mode,  External Interrupt. |
| 11000 | MUL X | ACC=ACC\*X | Multiply the value in the Accumulator with the Oprand. Then put the result back in the Accumulator. |
| 11001 | DIV X | ACC=ACC/X | Divide the value in the Accumulator with the Oprand. Then put the result back in the Accumulator. |
| 11010 | BRANCH X | ProCounter = X | Branch/Jump to the instruction address X. |
| 11011 | BRZERO X | If(ZeroFlag==1)  ProCounter =X | If zero flag is 1, then branch to the instruction address X. |
| 11100 | JUMPSUB X | SubAddrReg = ProCounter  ProCounter = X | Save the current Program Counter. Then jump to the subroutine. The subroutine address is provided in the Oprand. |
| 11101 | RETURNSUB | ProCounter = SubAddrReg | Return from the subroutine. This is done by loading the value from SubAddrReg into the Program Counter. |
| 11110 | RETURNITR | ProCounter = ItrAddrReg | Return from the interrupt service routine. This is done by loading the value from ItrAddrReg into the Program Counter. |

**Examples**

|  |  |
| --- | --- |
| Assembly Code | ADD @9 |
| Machine Code | 00000 001 00001001 (There will be no spaces in the real program) |
| Explanation | Add the value in the accumulator with the value at the Data Ram address location 9 |

|  |  |
| --- | --- |
| Assembly Code | ADD 3 |
| Machine Code | 00000 000 00000011 |
| Explanation | Add the value in the accumulator with 3. |

|  |  |
| --- | --- |
| Assembly Code | RETURNSUB |
| Machine Code | 11110 DDD DDDDDDDD ( ‘D’ means “Don’t Cares.” It can be 1 or 0, it won’t matter.) |
| Explanation | Put the value from the SubAddrReg into the program counter. |