**ECE 486/586**

**COMPUTER ARCHITECTURE**

**FINAL PROJECT REPORT**

**PDP8 ISA SIMULATOR**



**By**

**NileshDattani**

**PrathameshShete**

**Rohit Jadhav**

**Shalmalee Sahasrajbhojanee**

**Project Objective:**

For this course project, we have to design and simulate an entirepdp8 ISA simulatorcapable of generating memory trace file.

**Features of pdp8 minicomputer simulator:-**

* The PDP-8 is a 12-bit machine. It is Accumulator based architecture.
* With 12-bit addresses, up to 4096 words of memory can be addressed which is organized as 32 pages of 128 locations each.
* There are two registers in the PDP-8, two most commonly used registers are Accumulator register (12-bit) and the Link bit. There is also a 12-bit program counter.
* Main memory is accessed by two registers CPMA- which contains the address and MB – contains data.
* There are 4 addressing modes- Zeropage, current page , Indirect addressing, Autoindexing
* Input to the system is a .mem file with one new instruction encoding on every new line. Encoding is of 12 bits.
* At end of trace file complete statistics are displayed which include total instructions executed, total clock cycles required and number of instructions as per the mnemonic.

**Assumptions**:-

* This is merely simulator so it executes on the machine code generated by Pal assembler irrespective of whether instruction makes sense or not.
* PC is incremented only at one place. We set flag at other place wherever PC is changed.
* AC and Link bit is initialized to zero and memory is left uninitialized.
* Separate memread and memwrite tasks are written so that memory is accessed from one location only.
* I/O instructions, single stepping Breakpoint and Branch trace file are also implemented.

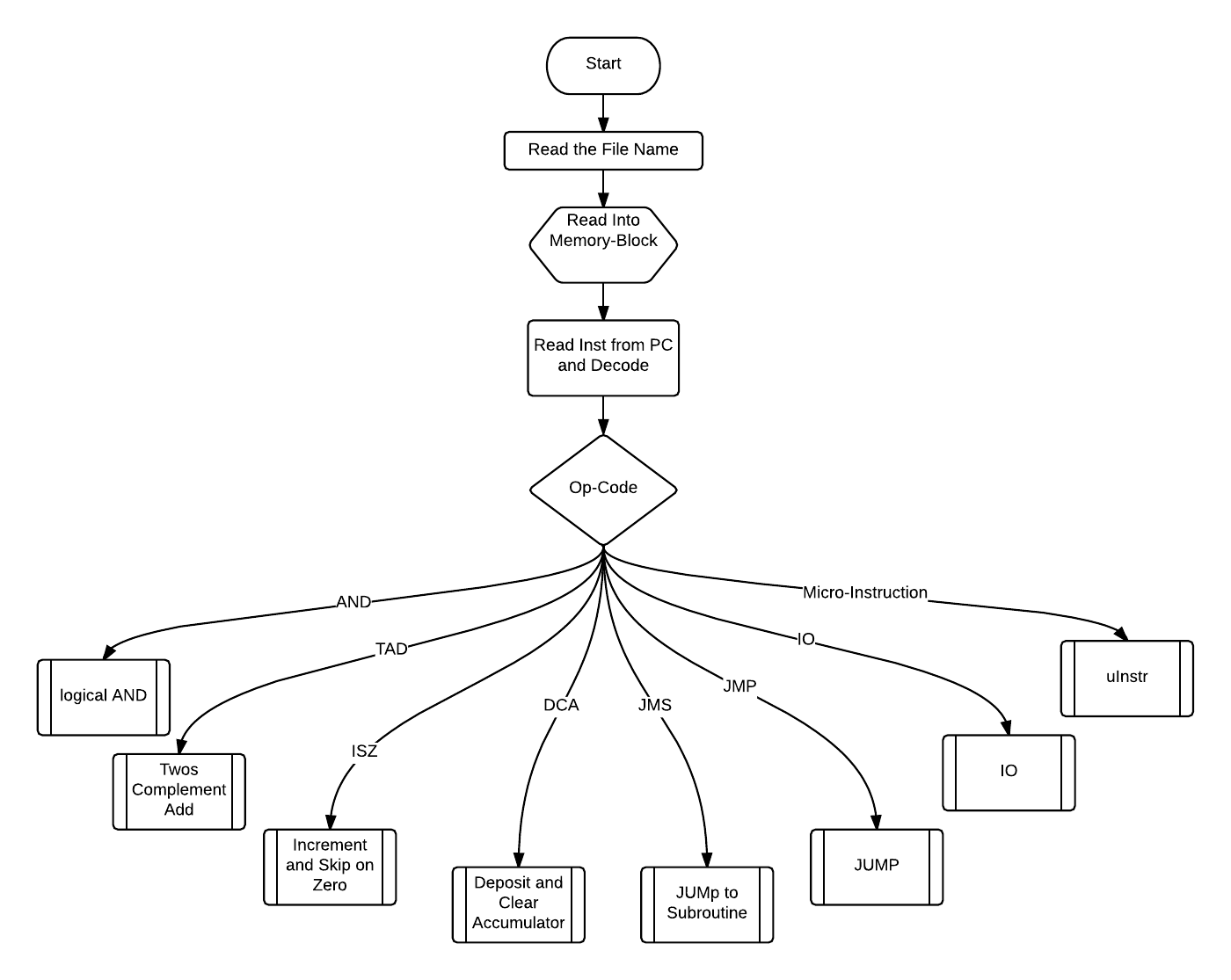
Note: - for I/O instructions, only single character is accepted at a time.

Number of simulated cycles for each instruction are as follows

|  |  |  |
| --- | --- | --- |
| **Mnemonic** | **Opcode** | **Cycles** |
| AND | 0 | 2 |
| TAD | 1 | 2 |
| ISZ | 2 | 2 |
| DCA | 3 | 2 |
| JMS | 4 | 2 |
| JMP | 5 | 1 |
| <IO> | 6 | 0 |
| µinstructions | 7 | 1 |

**Memory trace file**: - <type><address>

|  |  |
| --- | --- |
| **Type** | **Operation** |
| 0 | Data read |
| 1 | Data write |
| 2 | Instruction fetch |

**Block Diagram:-**The above block diagram represents the complete flow of the pdp8 simulator. The mem file (hex format) generated by pal assembler is read by ‘readmemh’function in run time. Instructions are then decoded and based on opcode instructions are executed. All statistics are recorded successfully.

**Instruction encoding format:-**

**Memory reference Instructions**

00 01 02 03 04 05 06 07 08 09 10 11

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

| | | | | | | | | | | | |

|\_\_|\_\_|\_\_|\_\_|\_\_|\_\_|\_\_|\_\_|\_\_|\_\_|\_\_|\_\_|

| | I| Z| 7 bit |

| 3 bit | | Word in Page |

| Opcode | Mode |

| | Address |

**Group 1 Microinstructions: - (bit 3=0)**

     0   1   2   3   4   5   6   7   8   9   10  11   
    +---+---+---+---+---+---+---+---+---+---+---+---+   
    | 1 | 1 | 1 | 0 |CLA|CLL|CMA|CML|RAR|RAL|0/1|IAC|   
    +---+---+---+---+---+---+---+---+---+---+---+---|

**Group 2 Microinstructions: - (bit 3=1, bit 11=0)**

  0   1   2   3   4   5   6   7   8   9   10 11   
    +---+---+---+---+---+---+---+---+---+---+---+---+   
    | 1 | 1 | 1 | 1 |CLA|SMA|SZA|SNL|0/1|OSR|HLT| 0 |   
    +---+---+---+---+---+---+---+---+---+---+---+---|

    Bit 8 : if 1 reverse logic to obtain SPA, SNA, SZL

**Group 2 Microinstructions: - (bit 3=1, bit 11=1)**

  0   1   2   3   4   5   6   7   8   9   10  11   
    +---+---+---+---+---+---+---+---+---+---+---+---+   
    | 1 | 1 | 1 | 1 |CLA|MQA| 0 |MQL| 0 | 0 | 0 | 1 |   
    +---+---+---+---+---+---+---+---+---+---+---+---|

**Functions(Tasks):-**

**A) Effective address:-**This is very important function and used for all memory reference instructions.

**B) Group1 and Group2 Microinstructions: -**Group 1 microinstructions are used to change the state of Link bit and accumulator. Group 2 instructions are conditional branch instructionswhich skips on certain states of AC and link bit.

**C) memread and memwrite: -**Memory access tasks are written separately so that memory accessed from only one place only removing any ambiguity.

**Testing Approach:-**

**Task level Testing:-** We tested our code at task level separately- Effective Address function, all the seven opcode instructions. Every single instruction was tested with separate assembly code. All the corner cases in terms of coverage and boundary were considered and tested successfully. All the tasks were tested for correctness, correct statistics in terms of number of clock cycles executed, number of instructions executed per opcode. Status of Accumulator and link bit was tested whenever required.

**Integration Testing:-**We thendeveloped regression suite of test cases i.e a big assembly language code which covers basic functionality of that particular task. This helped us to save time on testing every functionality every time whenever a defect was logged and corrected. Otherwise we would have to run every assembly code every time an update was made. All the statistics were checked after running regression suite of test cases.

**Extra credit Testing:-**Separate test cases were written for extra credit – branch trace file, I/O instructions and single stepping were tested separately successfully. .

**Ad-hoc testing: -**We then tested some critical test cases.We checked whether error messages were correctly displayed.We retested for overflow conditions for accumulator, link bit complementing while carry out, auto indexing modes, JMS and JMP instructions with special conditions.

Errors were detected and successfully removed. Comprehensive and exhaustive testing was carried out.

We have organized test cases as opcode/scenario, Test case name, assembly code name-for mapping, actual result, expected result & status. We have organized our test cases as per the functionality.

**Test Cases:- A) Memory Reference Instructions**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Opcode | TC description | Inst. Name | Assembly code name | expected result | Actual result | Status |
| 0 | anding memory contents with accumulator. | AND | and.txt | C=3;CLK=8 | C=3;CLK=8 | Pass |
| 0 | using AND in indirect mode | AND | indirect.txt | C=3;CLK=9 | C=3;CLK=9 | Pass |
| 0 | using AND in page zero mode | AND | zero page.txt | C=3;CLK=8 | C=3;CLK=8 | Pass |
| 0 | using AND in page zero and indirect mode | AND | zero\_indirect.txt | C=3;CLK=9 | C=3;CLK=9 | Pass |
|  |  |  |  |  |  | Pass |
| 1 | Adding 0 to accumulator. | TAD | tad\_adding\_0.txt | C=2; CLK=8 | C=2; CLK=8 | Pass |
| 1 | adding 2 positive numbers | TAD | tad\_adding2 num.txt | C=5; CLK=8 | C=5; CLK=8 | Pass |
| 1 | adding a negative number | TAD | tad\_adding\_-ve.txt | C=7775; CLK=8 | C=7775; CLK=8 | Pass |
| 1 | using ADD in indirect mode | TAD | tad\_indirect1.txt | C=3;CLK=9 | C=3;CLK=9 | Pass |
| 1 | using indirect mode to access page zero | TAD | tad\_indirect2.txt | C=3;CLK=9 | C=3;CLK=9 | Pass |
| 1 | using ADD in page zero mode | TAD | tad\_zero page.txt | C=3;CLK=8 | C=3;CLK=8 | Pass |
| 1 | using ADD in page zero and indirect mode | TAD | tad\_zero\_indirect.txt | C=5; CLK=9 | C=5; CLK=9 | Pass |
| link bit; 1 | to chk if link bit flips properly | TAD | tad\_link\_bit test.txt | link=0;clk=10 | link=0;clk=10 | Pass |
| overflow;1 | to chk if overflow occurs by adding +ve nos. | TAD | tad\_overflow.txt | overflow occurred | overflow occurred | Pass |
| overflow;1 | to chk if overflow occurs by adding -ve nos. | TAD | tad\_overflow1.txt | overflow occurred | overflow occurred | Pass |
|  |  |  |  |  |  | Pass |
| 2 | tochk if nxt instruction skips. | ISZ | ISZ\_+ve.txt | instruction skips | instruction skips | Pass |
| 2 | tochk if nxt instruction does not skips. | ISZ | ISZ\_-ve.txt | instruction does not skip | instruction does not skip | Pass |
| 2 | using ISZ in indirect mode +ve test | ISZ | isz\_indirt+ve.txt | instruction skips | instruction skips | Pass |
| 2 | using ISZ in indirect mode -ve test | ISZ | isz\_indirt-ve.txt | instruction does not skip | instruction does not skip | Pass |
| 2 | using ISZ in zero page mode +ve test | ISZ | ISZ\_zeroPage+ve .txt | instruction skips | instruction skips | Pass |
| 2 | using ISZ in zero page mode -ve test | ISZ | isz\_zero-ve.txt | instruction does not skip | instruction does not skip | Pass |
| 2 | using ISZ in zero page and indirect mode +ve test | ISZ | isz\_zero\_indirt+ve.txt | instruction skips | instruction skips | Pass |
| 3 | checking functioning of DCA | DCA | dca.txt | C=5; CLK=8 | C=5; CLK=8 | Pass |
| 3 | using DCA in indirect mode | DCA | dca\_indirect.txt | D=5; CLK=9 | D=5; CLK=9 | Pass |
| 3 | using DCA in page zero mode | DCA | dca\_zeropage.txt | C=5; CLK=8 | C=5; CLK=8 | Pass |
| 3 | using DCA in page zero and indirect mode | DCA | dca\_zero\_indirect.txt | D=5; CLK=9 | D=5; CLK=9 | Pass |
|  |  |  |  |  |  | Pass |
| 4 | checking functionality of JMS | JMS | jms.txt | C=5; CLK=12 | C=5; CLK=12 | Pass |
| 4 | using JMS in indirect mode | JMS | jmsi.txt | C=5; CLK=13 | C=5; CLK=13 | Pass |
| 4 | using JMS in page zero mode | JMS | jmsz.txt | C=5; CLK=13 | C=5; CLK=13 | Pass |
| 4 | using JMS in page zero and indirect mode | JMS | jmszi.txt | C=5; CLK=13 | C=5; CLK=13 | Pass |
|  |  |  |  |  |  | Pass |
| 5 | checking functionality of JMP | JMP | jmp.txt | D=5; CLK=9 | D=5; CLK=9 | Pass |
| 5 | using JMP in indirect mode | JMP | jmpi.txt | D=5; CLK=10 | D=5; CLK=10 | Pass |
| 5 | using JMP in page zero mode | JMP | jmpz.txt | D=5; CLK=11 | D=5; CLK=11 | Pass |
| 5 | using JMP in page zero and indirect mode | JMP | jmpzi.txt | D=5; CLK=10 | D=5; CLK=10 | Pass |
| 5 | jumping to different page | JMP | jumping to other page.txt | D=5; CLK=10 | D=5; CLK=10 | Pass |
|  |  |  |  |  |  | Pass |
| Loop | Executing a loop until the data element is zero | loop | fun1.txt | sum=136; CLK=48 | sum=136; CLK=48 | Pass |
| function | passing values to subroutine to add | subroutine | fun2.txt | C=7; CLK=18 | C=7; CLK=18 | Pass |
| start address | start address other than 200 octal | - | start\_address.txt | C=5; CLK=8 | C=5; CLK=8 | Pass |

**B.Group2 microinstructions (SubGroup AND)**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| TC description | Inst. Name | Assembly code name | expected result | Actual result | Result |
| To verify whether instruction is skipped upon SZL(LB=0) | SZL | [SZL.as](http://SZL.as) | TAD B should be skipped.AC=2 | TAD B should is skipped.AC=2 | Pass |
| To verify whether instruction is not skipped upon SZL(LB=1) | SZL | [SZL\_neg.as](http://SZL_neg.as) | TAD B should not be skipped | TAD B should not be skipped | Pass |
| To verify whether instruction is not skipped for SNA when AC=0 | SNA | [SNA\_0.as](http://SNA_0.as) | TAD B should not be skipped.AC=47 | TAD B is not be skipped.AC=47 | Pass |
| To verify whether instruction is not skipped for SNA when AC=0 | SNA | [SNA\_negetive.as](http://SNA_negetive.as) | TAD B should be skipped. AC=-5 | TAD B is skipped. AC=-5 | Pass |
| To verify whether instruction is not skipped for SNA when AC=+ve | SNA | [SNA\_positive.as](http://SNA_positive.as) | TAD B should be skipped.AC=5 | TAD B is skipped. AC=-5 | pass |
| To verify whether instruction is skipped upon SPA(AC=+ve) | SPA | [SPA\_positive.as](http://SPA_positive.as) | TAD B should be skipped.AC=5 | TAD B is skipped.AC=5 | Pass |
| To verify whether instruction is skipped upon SPA(AC=0) | SPA | [SPA\_0.as](http://SPA_0.as) | TAD B should be skipped.AC=45 | TAD B is skipped.AC=45 | Pass |
| To verify whether instruction is not skipped upon SPA(AC=-ve) | SPA | [SPA\_neg.as](http://SPA_neg.as) | TAD B should not be skipped.AC=-3 | TAD B is not skipped.AC=-3 | pass |
| To verify whether instruction is skipped upon SNA\_SZL(AC=+ve, LB=0) | SNA\_SZL | [SNA\_SZL.as](http://SNA_SZL.as) | TAD B should be skipped.AC=5 | TAD B is skipped.AC=5 | Pass |
| To verify whether instruction is not skipped when one of the condition is not satisfied(AC=0) | SNA\_SZL | [SNA\_SZL\_AC0.as](http://SNA_SZL_AC0.as) | TAD B should not be skipped.AC=2 | TAD B should is not skipped.AC=2 | Pass |
| To verify whether instruction is not skipped when one of the condition is not satisfied(LB=1) | SNA\_SZL | [SNA\_SZL\_LB1.as](http://SNA_SZL_LB1.as) | TAD B should not be skipped.AC=7 | TAD B should is not skipped.AC=7 | Pass |
| To verify whether instruction is skipped upon SNA\_SPA(AC=+ve) | SPA\_SNA | [SPA\_SNA.as](http://SPA_SNA.as) | TAD B should be skipped.AC=5 | TAD B is skipped.AC=5 | Pass |
| To verify whether instruction is not skipped when one of the condition is not satisfied(AC=-ve) | SPA\_SNA | [SPA\_SNA\_negAC.as](http://SPA_SNA_negAC.as) | TAD B should not be skipped.AC=-3 | TAD B should is not skipped.AC=-3 | Pass |
| To verify whether instruction is skipped upon SNA\_SPA and SZL(AC=+ve& LB=0) | SPA\_SNA\_SZL | [SPA\_SNA\_SZL.as](http://SPA_SNA_SZL.as) | TAD B should be skipped.AC=5 | TAD B is skipped.AC=5 | Pass |
| To verify whether instruction is not skipped upon SNA\_SPA and SZL(AC=0 & LB=0). one of the condition is not satisfied | SPA\_SNA\_SZL | [SPA\_SNA\_SZL\_neg.as](http://SPA_SNA_SZL_neg.as) | TAD B should not be skipped.AC=7 | TAD B is not skipped.AC=7 | Pass |

**Group2 microinstructions (Subgroup OR)**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| TC description | Inst. Name | Assembly code name | expected result | Actual result | Status |
| To verify whether instruction is skipped when Accumulator is negative(AC=4015 i.e MSB=1) | SMA (Skip on Minus AC) | SMA.as | TAD B be should be skipped and AC=4015 | TAD B is not skipped. ACC=4017 | Pass |
| To verify whether instruction is skipped when Accumulator is zero | SZA (Skip on Zero AC) | [SZA.as](http://SZA.as) | TAD B should be skipped. AC=1 | TAD B is skipped. AC=1 | pass |
| All 3 conditions are not satisfied. So next instruction should not be skipped | SMA\_SZA\_SNL | [SMA\_SZA\_SNL\_neg.as](http://SMA_SZA_SNL_neg.as) | TAD B should not be skipped | TAD B is not skipped. AC=4 | Pass |
| To verify whether Skip is performed for the combination negative accumulator and non-zero link. | SZA\_SNL\_ACneg | [SZA\_SNL\_ACneg.as](http://SZA_SNL_ACneg.as) | TAD B should skipped | TAD B is skipped | Pass |
| To verify whether skip is not performed when AC is nonzero for SZA instruction | SZA\_neg | [SZA\_neg.as](http://SZA_neg.as) | TAD B should not be skipped | TAD B is not skipped | Pass |
| To verify whether skip is performed when AC is zero for SZA instruction | SZA | [SZA.as](http://SZA.as) | TAD B should be skipped. | TAD B is skipped. | Pass |
| To verify whether skip is performed when only one of the condition of OR subgroup is true.(AC=0, LB=0) | SZA\_SNL | [SZA\_snl\_LB0.as](http://SZA_snl_LB0.as) | TAD B should be skipped. | TAD B is skipped. | Pass |
| To verify whether skip is not performed when both of the condition of OR subgroup are false.(AC=+ve, LB=0) | SZA\_SNL | SZA\_SNL\_neg | TAD B should not be skipped. | TAD B is not skipped. | Pass |
| To verify whether skip is performed when only one of the condition of OR subgroup is true.(AC=-ve, LB=0) | SMA\_SNL | [SMA\_SNL\_ACneg.as](http://SMA_SNL_ACneg.as) | TAD B should be skipped. | TAD B is skipped. | Pass |
| To verify whether skip is performed when only one of the condition of OR subgroup is true.(AC=+ve, LB=1) | SMA\_SNL | [SMA\_SNL\_LB1.as](http://SMA_SNL_LB1.as) | TAD B should be skipped. AC=33 | TAD B is skipped. AC=33 | Pass |
| To verify whether skip is not performed when both of the condition of OR subgroup are not satisfied.(AC=+ve, LB=0) | SMA\_SNL | [SMA\_SNL\_neg.as](http://SMA_SNL_neg.as) | TAD B should not be skipped. | TAD B is not skipped. | Pass |
| To verify whether skip is performed when only one of the condition of OR subgroup is true.(AC=-ve) | SMA\_SZA | [SMA\_SZA.as](http://SMA_SZA.as) | TAD B should be skipped.AC=4017 | TAD B is skipped. AC=4017 | Pass |
| To verify whether skip is performed when only one of the condition OR group is true.(AC=0) | SMA\_SZA | [SMA\_SZA\_neg.as](http://SMA_SZA_neg.as) | TAD B should be skipped.AC=1 | TAD B is skipped. AC=1 | PASS |
| To verify whether skip is not performed when accumulator is not negative(AC=+ve) | SMA | [SMA\_neg.as](http://SMA_neg.as) | TAD B should not be skipped. AC=17 | TAD B should is not skipped. AC=17 | Pass |
| To verify whether skip is not performed when LinkBit is zero(LB=0) | SNL | [SNL\_neg.as](http://SNL_neg.as) | TAD B should not be skipped. AC=3 | TAD B should is not skipped. AC=3 | Pass |
| To verify whether skip is not performed when LinkBit is one(LB=1) | SNL | [SNL.as](http://SNL.as) | TAD B should be skipped. AC=1 | TAD B is skipped. AC=1 |  |
| To verify operation of SKP | SKP (Skip Always) | [SKP.as](http://SKP.as) | C=45 and not 47. | AC=45 | Pass |
| To verify behaviour for SNL and SKP | SNL\_SKP | [SNL\_SKP.as](http://SNL_SKP.as) | It should behave as SZL | It behaves as SZL | Pass |
| To verify behavior of combined instruction of SZA CLA | SZA\_CLA | [SZA\_CLA.as](http://SZA_CLA.as) | Next instruction should be skipped and AC is cleared. | Next instruction is skipped and AC is cleared. | Pass |
| To verify behavior of combined instruction of SMA CLA | SMA\_CLA | [SMA\_CLA.as](http://SMA_CLA.as) | Next instruction should be skipped and AC is cleared. | Next instruction is skipped and AC is cleared. | Pass |
| To verify behavior of combined instruction of SNL CLA | SNL\_CLA | SNL\_CLA | Next instruction should be skipped and AC is cleared. | Next instruction is skipped and AC is cleared. | Pass |

**Group2 – Other Instructions and Combinations with AND/OR subgroup**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| TC description | Inst. Name | Assembly code name | expected result | Actual result | Result |
| To verify the combination of SNA & CLA | SNA\_CLA | [SNA\_CLA.as](http://SNA_CLA.as) | TAD C should be skipped and AC should be zero | TAD C is skipped and AC is zero. | PASS |
| To verify the PC value after last SKP instruction. | SNA\_CLA | [SNA\_CLA.as](http://SNA_CLA.as) | PC value should be 206 | PC value is 206 | PASS |
| To verify behavior when HLT instruction is skipped | SKP\_HLT | [SKP\_HLT.as](http://SKP_HLT.as) | HLT should be skipped, so PC goes till PC location 4095. | HLT should be skipped, so PC goes till PC location 4095. | PASS |
| To verify behavior for combined instructions SNL & SKP | SNL\_SKP | [SNL\_SKP.as](http://SNL_SKP.as) | It should behave as SZL. PAL produces instruction encoding of SZL | It behaves as SZL | PASS |
| To verify behavior of combined instruction of SZA CLA | SZA\_CLA | [SZA\_CLA.as](http://SZA_CLA.as) | Next instruction should be skipped and AC is cleared. | Next instruction is skipped and AC is cleared. | PASS |
| To verify behavior of combined instruction of SMA CLA | SMA\_CLA | [SMA\_CLA.as](http://SMA_CLA.as) | Next instruction should be skipped and AC is cleared. | Next instruction is skipped and AC is cleared. | PASS |
| To verify behavior of combined instruction of SNL CLA | SNL\_CLA | SNL\_CLA | Next instruction should be skipped and AC is cleared. | Next instruction is skipped and AC is cleared. | PASS |
| To verify behavior of combined instruction of CLA | CLA (Clear AC) |  | AC should be cleared. | AC is cleared. | PASS |
| To verify whether instructions HLT halts the execution program | HLT (Halt) |  | Instructions should stop executing afterwards | No instructions is executed after HLT | PASS |
| To verify whether all basic group 2 instructions are executed.(Regression Suite TC) | All\_Group2 | All\_Group2 | CLK=74. All instructions should be executed successfully. | All Instructions are successfully | PASS |

**D. Group 3 Instructions**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| TC description | Inst. Name | Assembly code name | expected result | Actual result | status |
| To verify the working of all group 3 instructions | CLA, MQL,MQA,SWP,CAM | [grp3.as](http://grp3.as) | All the instructiosns should be executed successfully. | All the group 3 instructisn are executed successfully. | Pass |

**E. Effective Addressing**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| TC description | Inst. Name | Assembly code name | expected result | Actual result | status |
| To verify operation AutoIndexing operation when Autoindexing locations are accessed with or without autoindexing modes. | Autoindex (without & with I bit) | [AutoIndex\_withoutIndirect.as](http://AutoIndex_withoutIndirect.as) | The one final AC should be 204. Test the output.txt file number of read and write operations. CLK= 14. For Autoindexing locations with i bit set data read followed by data write should happen. | Final AC is 204.Output.txt file verified for correct number of reads and writes. CLK=14.For Autoindexing locations with i bit set data read followed by data write is happening. | Pass |
| To verify all the combinations of Addressing Modes- Direct Zero, Direct Current, Indirect Zero, Indirect current | All addressing modes | [All\_addressingModes.as](http://All_addressingModes.as) | All addressing modes should work properly. CLK=23 | All addressing modes works properly.CLK=23 | Pass |

**F. Extra Credit**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Inst. Name | Assembly code name | expected result | Actual result | Result |
| Loop | fun1.txt | All branches should be displayed correctly. sum=136; CLK=48 | All branches are displayed correctly. sum=136; CLK=48 | Pass |
| KRB, TLS | [IO1.as](http://IO1.as) | A correct ASCII representation of the letter entered | All letters (Uppercase/ Lowercase) and symbols displayed correctly | Pass |
| Single stepping |  | Should run through the program completely and display just summary if single stepping not chosen else display current instruction encountered, PC value, link value, ACC value and statement "Press key to continue" | On pressing y/n, correctly single steps depending on choice through the program. On pressing any key, if single step chosen, gives step by step display of the contents. | Pass |