Lab/Tutorial 4
Changing Condition Codes and Displaying Stages
Design, Implementation and Testing Document

Prepared for: Dr. Larry Hughes

Abdulla Sadoun B00900541

Table of Contents

Problem Introduction

Statement of Purpose

The purpose of this laboratory part(1) is to alter the current instruction set in the emulator to implement instructions from the isa that are responsible for easily altering PSW or program status register. We will implement the fetch, decode and execute of the function or instruction SETCC which sets specified bits on the PSW register and we will also implement the same stages for CLRCC which clears specified bits on the PSW.

On the second Part (2) of this laboratory, we will implement a new view for the debugger when it goes to execute the code in debugger mode. 2 new functions will be introduced:

- process _instructions_debugger(): This function will run all the instructions in instruction-memory (IMEM) until the end of the instructions or until a breakpoint is encountered. It will have a more debugger friendly interface, showing the different stages it is going through and the time counter incrementing.
- step_debugger(): This function will run a single cpu clock tick, and display a similar result and view to the process_instructions_debugger() function, but this time it will do it in single steps when the user calls for this option. This function will be a great asset and tool when it comes to debugging code being executed on the CPU using our XM23p Emulator.

The design, implementation and testing of this CPU Emulator will aid their students in their journey to learning the depths of a CPU's architecture to gain the knowledge necessary to complete the required course: Computer Architecture. The course stands as a necessary asset for the computer engineers the students aspire to become.

Design:

Data Dictionary

Registers[RegisterNo][BitNo] = [General Purpose Registers|Special Purpose Registers],[Bit Number]
General Purpose Register = [R0|R1|R2|R3|R4]
Special Purpose Registers = [PC|SP|LR]
R0 = [‘0’| ‘000’]
R1 = [‘1’| ‘001’]
R2 = [‘2’| ‘010’]

R3 = [‘3’| ‘011’]
R4 = [‘4’| ‘100’]
LR = 5
SP = 6
PC = 7

s-record = ‘s’ + type + length of record + Address + Data + Checksum
Type = [0|1|2|9]
Length of record = Byte Pair
Address = 2[Byte Pair]2
Address = 0000-ffff
Data = 1[Byte Pair]30
Byte Pair = character + character
Character = [0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F]
Checksum = 1[Byte Pair]1

General Instruction = Opcode + Operand
Opcode = 4{bit}13
Bit = [0|1]

Operand = [RC|WB|Source|Destination|Byte]
RC = [Register|Constant]
Register = 0
Constant = 1

WB = [Word|byte]
Word = 2{byte}2
Byte = 8{bit}8

Source = [R0-R4] *in bits*
Destination = [R0-R4] *int bits*

Register Instructions (ADD-SXT) = Opcode + Operand

Register Initialization Instructions (MOVL-MOVH) = Opcode + Operand

Breakpoint = IMEMaddress
IMEMaddress = Address *Address in Instruction memory to stop executing*
Address = 2[Byte Pair]2

PSW instructions = 1[CLRCC_Opcode|SETCC_Opcode]1 + PSW_Operands
SETCC_Opcode = “0” + “1” + “0” + “0” + “1” + “1” + “0” + “1” + “1” + “0” + “1”
CLRCC_Opcode = “0” + “1” + “0” + “0” + “1” + “1” + “0” + “1” + “1” + “1” + “0”

PSW_Operands = Overflow_Bit + Negative_Bit + Zero_bit + Carry_bit
Overflow_Bit = [0|1]

Negative_Bit = [0|1]
Zero_bit = [0|1]
Carry_bit = [0|1]

Pseudo Code

Main.c

There are no changes being done to the main in this laboratory.

Decode.c

```
FUNCTION Decode:
        ENUM {BLCase, BEQtoBRA, ADDtoST, MOVLtoMOVH, LLDR, LSTR}
        IF (E_Start_Addresses - 2) equals BreakpointValue
           PRINT "Breakpoint reached3"
           RETURN Error
        END IF

        IF (RegistersValue[PC]) equals BreakpointValue
           PRINT "Breakpoint reached4"
           RETURN Error
        END IF

        SET opcode = (IMARValue >> 13) AND 0x07
        SET off = (IMARValue >> 7) AND 0x7f

        SWITCH (opcode)
           CASE BLCase:
              RETURN BL
           CASE BEQtoBRA:
              RETURN betweenBEQandBRA(IMARValue)
           CASE ADDtoST:
              RETURN betweenADDandST(IMARValue)
           CASE MOVLtoMOVH:
              RETURN betweenMOVLandMOVH(IMARValue)
           CASE LLDR:
              RETURN LDR
           CASE LSTR:
              SET offbuff = off
```

```
            RETURN STR
        DEFAULT:
            SET offbuff = off
            PRINT "Error - instruction not yet implemented"
            RETURN Error
    END SWITCH
END FUNCTION
```

FUNCTION betweenADDandST (IMARValue):

```
    *Set Layer one opcode and buffers* - unchanged
    IF opcode equals LTCASE OR opcode equals STCASE
        SET opcode = (IMARValue >> 10) AND 0x01
        SET prpobuff = (IMARValue >> 9) AND 0x01
        SET decbuff = (IMARValue >> 8) AND 0x01
        SET incbuff = (IMARValue >> 7) AND 0x01
        SET wbbuff = (IMARValue >> 6) AND 0x01
        SET srcbuff = (IMARValue >> 3) AND 0x07
        SET dstbuff = IMARValue AND 0x07

        IF opcode equals SUB_LD
            RETURN LD
        ELSE
            RETURN ST
        END IF

    ELSE IF opcode equals MOV_CLRCC
        ENUM {LMOV, LSWAP, LSRALSXT, LSETPRILCLRCC, LSVC, LSETCC, LCLRCC}
        SET opcode = (IMARValue >> 7) AND 0x07

        SET vbuff = (IMARValue >> 4) AND 0x01
        SET nbuff = (IMARValue >> 2) AND 0x01
        SET zbuff = (IMARValue >> 1) AND 0x01
        SET cbuff = IMARValue AND 0x01

        SWITCH (opcode)
            CASE LMOV:
                RETURN MOV
            CASE LSWAP:
                RETURN SWAP
            CASE LSRALSXT:
                SET opcode = (IMARValue >> 3) AND 0x07
                ENUM {LSRA, LRRC, LSWPB, LSXT}
                SWITCH (opcode)
            CASE LSRA:
                RETURN SRA
            CASE LRRC:
                RETURN RRC
            CASE LSWPB:
                RETURN SWPB
            CASE LSXT:
                RETURN SXT
            DEFAULT:
                PRINT "instruction not yet implemented"
                RETURN Error
        END SWITCH
    CASE LSETPRILCLRCC:
        SET opcode = (IMARValue >> 4) AND 0xf
        IF opcode equals 8
            RETURN SETPRI
        ELSE IF opcode equals 9
            RETURN SVC
        ELSE IF opcode equals 10 OR opcode equals 11
            RETURN SETCC
        ELSE IF opcode equals 12 OR opcode <= 13
            RETURN CLRCC
        ELSE
            PRINT "instruction not yet implemented"
            RETURN Error
        END IF
    DEFAULT:
        PRINT "instruction not yet implemented"
        RETURN Error
END SWITCH

ELSE IF opcode equals CEX
    RETURN CEX

ELSE
    SET rcbuff = (IMARValue >> 7) AND 0x01
    SET wbbuff = (IMARValue >> 6) AND 0x01
    SET srcbuff = (IMARValue >> 3) AND 0x07
    SET dstbuff = IMARValue AND 0x07
    SET opcode = (IMARValue >> 8) AND 0x0F

    ENUM {LADD, LADDC, LSUB, LSUBC, LDADD, LCMP, LXOR, LAND, LOR, LBIT,
LBIC, LBIS}
```

```
        SWITCH (opcode)
            CASE LADD:
                RETURN ADD
            CASE LADDC:
                RETURN ADDC
            CASE LSUB:
                RETURN SUB
            CASE LSUBC:
                RETURN SUBC
            CASE LDADD:
                RETURN DADD
            CASE LCMP:
                RETURN CMP
            CASE LXOR:
                RETURN XOR
            CASE LAND:
                RETURN AND
            CASE LOR:
                RETURN OR
            CASE LBIT:
                RETURN BIT
            CASE LBIC:
                RETURN BIC
            CASE LBIS:
                RETURN BIS
            DEFAULT:
                PRINT "instruction not yet implemented"
                RETURN Error
        END SWITCH
    END IF


END FUNCTION
```

The rest of the file is unchanged.

Execute.c (changes for SETCC and CLRCC)

- Include Header file

```
FUNCTION execute(int instruction number)
    SWITCH (instruction number)
        CASE BL:
            # execution for BL
        BREAK
    CASE “INSTRUCTION”:
      # execution for “INSTRUCTION”
        BREAK
    CASE SETCC:
        PRINT "SETCC:"
        IF vbuff equals TRUE
              SET PSW.v = TRUE
        END IF
        IF nbuff equals TRUE
              SET PSW.n = TRUE
        END IF
        IF zbuff equals TRUE
              SET PSW.z = TRUE
        END IF
        IF cbuff equals TRUE
              SET PSW.c = TRUE
        END IF
        BREAK
    CASE CLRCC:
      PRINT "CLRCC:"
      IF vbuff equals TRUE
        SET PSW.v = FALSE
      END IF
      IF nbuff equals TRUE
        SET PSW.n = FALSE
      END IF
      IF zbuff equals TRUE
        SET PSW.z = FALSE
      END IF
      IF cbuff equals TRUE
        SET PSW.c = FALSE
      END IF
      BREAK



    CASE Error:
      PRINT "Error - instruction not yet implemented"
        BREAK
    DEFAULT:
      PRINT "Error - instruction not yet implemented"
```

```
            BREAK
        RETURN
    END SWITCH
END FUNCTION
```

Debugger.c

All unchanged except for the new function introduced to run in display mode

```
FUNCTION process_instruction_debugger:
      PRINT "Clock\tPC\tInstruction\tFetch\t\tDecode\tExecute"
      SET firstrun = TRUE
      SET D0 = 0
      SET E0 = 0
      WHILE (TRUE)
        Call function fetch
        INCREMENT I_Start_Addresses by 4
        INCREMENT E_Start_Addresses by 2
        SET RegistersValue[PC] = E_Start_Addresses
        Call function ChangedRegistersValue with (RegistersValue[PC], PC)
        SET F0 = RegistersValue[PC] - 2
        SET F1 = IMARValue

        IF IMARValue equals 0000 OR RegistersValue[PC] - 2 equals BreakpointValue
          PRINT "Breakpoint reached or the end of exec."
          BREAK
        END IF

        SET instructionnumber = Call function decode

        PRINT timecounter, RegistersValue[PC] - 2, IMARValue, "F0:", F0, "D0:", D0

        INCREMENT timecounter

        PRINT timecounter, "F1:", F1, "E0:", E0

        IF firstrun equals TRUE
          SET buffer = instructionnumber
          PRINT newline
          Call function fetch
          SET D0 = F1
          SET E0 = D0
          SET firstrun = FALSE
          INCREMENT timecounter
        ELSE
```

```
          Call function execute with (buffer)
          Call function fetch
          SET D0 = F1
          SET E0 = D0
          SET buffer = instructionnumber
          INCREMENT timecounter
        END IF
      END WHILE
END FUNCTION
```

How to use/run the software:

To run the program, since it is written entirely in C, any machine with a gcc/gnu compiler can be used in any machine.
First ensure you have all the files in one directory or folder to be able to run this program, the files are the loader.h file, loader.c and main.c files. Navigate to that directory using the terminal using “cd <directory>”. Once in the correct directory use “gcc -o main main.c debugger.c decode.c execute.c fetch.c loader.c xm23p.c” to compile the program and create the “.o” file named loader, now run loader using the following command “./emulator”. You should see the command window pop up and you would be able to use the menu to perform different functions. To enter the debugger mode and test the new functions, select the “d” option.

Implementation:

Included in the submission zip.
Changes mainly made in decode.c, debugger.c and execute.c

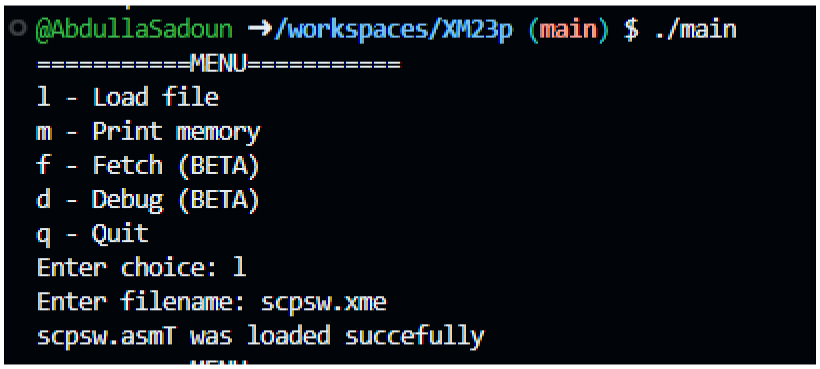
Testing

Test 1: Testing SETCC instruction

Purpose/Objective: The purpose of this test is to check if the software’s new PSW set instruction function works properly and performs as it should setting different values in the Program Status Register.
Test Configuration: For this test, I ran a program that utilizes the SETCC function and I will check the values of the PSW register before and after executing the SETCC instruction.

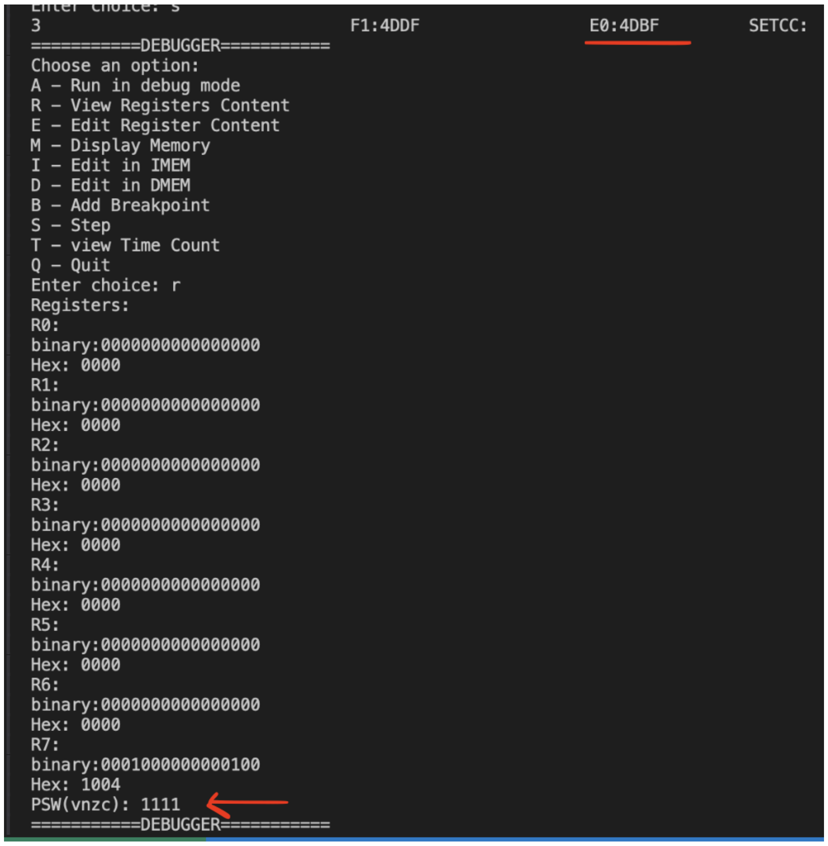
```
X-Makina Assembler - Version XM-23P Single Pass+ Assembler - Release 24.04.17
Input file name: scpsw.asm
Time of assembly: Tue 9 Jul 2024 04:56:08
 1                                     org #1000
 2                       MAIN
```

```
3       1000    4DBF              setcc     vsnzc
4       1002    4DDF              clrcc     vsnzc
5       1004    4DB1              setcc   vc
6       1006    4DD1              clrcc   vc
7                         DONE
8       1008    3FFF              bra                 DONE                        ; loop
9                         end MAIN
Successful completion of assembly - 1P

** Symbol table **

Constants (Equates)
Name                      Type      Value     Decimal

Labels (Code)
Name                      Type      Value     Decimal
DONE                      REL       1008      4104      PRI
MAIN                      REL       1000      4096      PRI

Labels (Data)
Name                      Type      Value     Decimal

Registers
Name                      Type      Value     Decimal
R7                        REG       0007      7         PRI
R6                        REG       0006      6         PRI
R5                        REG       0005      5         PRI
R4                        REG       0004      4         PRI
R3                        REG       0003      3         PRI
R2                        REG       0002      2         PRI
R1                        REG       0001      1         PRI
R0                        REG       0000      0         PRI

.XME file: \\Mac\Home\Desktop\Computer Architecture\Assembler\scpsw.xme
```



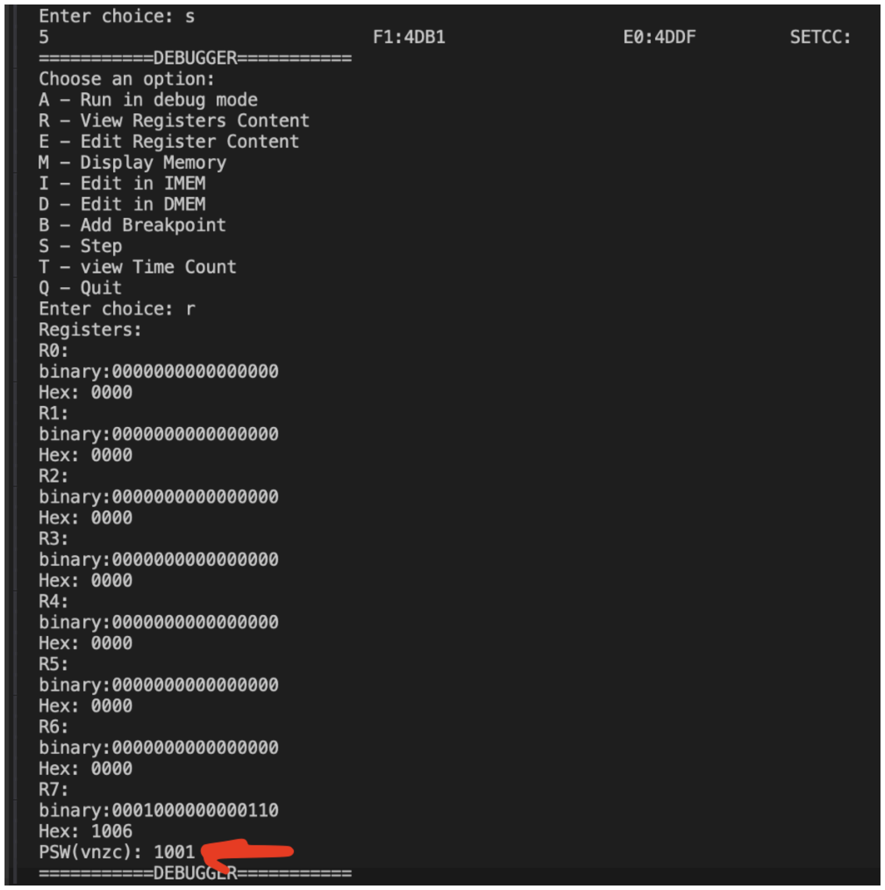
```
@AbdullaSadoun →/workspaces/XM23p (main) $ ./main
===========MENU===========
l - Load file
m - Print memory
f - Fetch (BETA)
d - Debug (BETA)
q - Quit
Enter choice: l
Enter filename: scpsw.xme
scpsw.asmT was loaded succefully
```

Expected Results: The newly implemented SETCC should work as expected setting all the fields then only the vc fields in the second call in the PSW register.
Actual Results: The actual result was as expected and the instruction ran as it should setting all the bits in the PSW register in the first run, and only the vc bits in the second run



```
Enter choice: s
3                            F1:4DDF                   E0:4DBF        SETCC:
===========DEBUGGER===========
Choose an option:
A - Run in debug mode
R - View Registers Content
E - Edit Register Content
M - Display Memory
I - Edit in IMEM
D - Edit in DMEM
B - Add Breakpoint
S - Step
T - view Time Count
Q - Quit
Enter choice: r
Registers:
R0:
binary:0000000000000000
Hex: 0000
R1:
binary:0000000000000000
Hex: 0000
R2:
binary:0000000000000000
Hex: 0000
R3:
binary:0000000000000000
Hex: 0000
R4:
binary:0000000000000000
Hex: 0000
R5:
binary:0000000000000000
Hex: 0000
R6:
binary:0000000000000000
Hex: 0000
R7:
binary:0001000000000100
Hex: 1004
PSW(vnzc): 1111
===========DEBUGGER===========
```

First call of SETCC



```
Enter choice: s
5                               F1:4DB1                 E0:4DDF         SETCC:
===========DEBUGGER===========
Choose an option:
A - Run in debug mode
R - View Registers Content
E - Edit Register Content
M - Display Memory
I - Edit in IMEM
D - Edit in DMEM
B - Add Breakpoint
S - Step
T - view Time Count
Q - Quit
Enter choice: r
Registers:
R0:
binary:0000000000000000
Hex: 0000
R1:
binary:0000000000000000
Hex: 0000
R2:
binary:0000000000000000
Hex: 0000
R3:
binary:0000000000000000
Hex: 0000
R4:
binary:0000000000000000
Hex: 0000
R5:
binary:0000000000000000
Hex: 0000
R6:
binary:0000000000000000
Hex: 0000
R7:
binary:0001000000000110
Hex: 1006
PSW(vnzc): 1001
===========DEBUGGER===========
```

Second call of SETCC

Pass/Fail: Pass

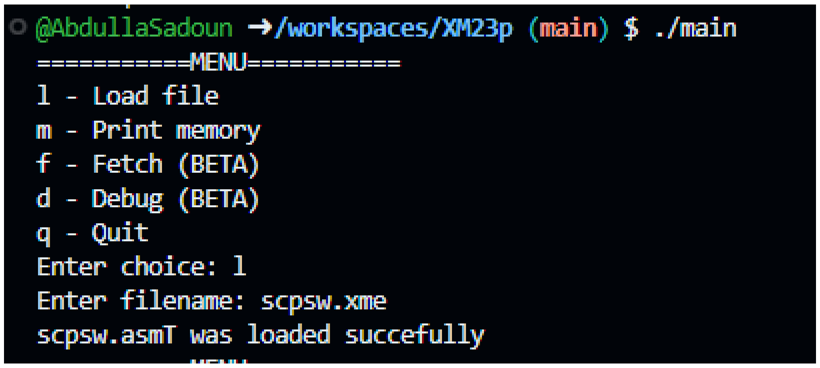
Test 2: Testing CLRCC instruction

Purpose/Objective: The purpose of this test is to check if the software's new PSW clear instruction function works properly and performs as it should clearing different values in the Program Status Register.

Test Configuration: For this test, I ran a program that utilizes the CLRCC function and I will check the values of the PSW register before and after executing the CLRCC instruction.

```
X-Makina Assembler - Version XM-23P Single Pass+ Assembler - Release 24.04.17
Input file name: scpsw.asm
Time of assembly: Tue 9 Jul 2024 04:56:08
 1                                  org #1000
 2                      MAIN
 3      1000    4DBF            setcc     vsnzc
 4      1002    4DDF            clrcc     vsnzc
 5      1004    4DB1       setcc  vc
 6      1006    4DD1            clrcc   vc
 7                      DONE
 8      1008    3FFF            bra             DONE                    ; loop
```

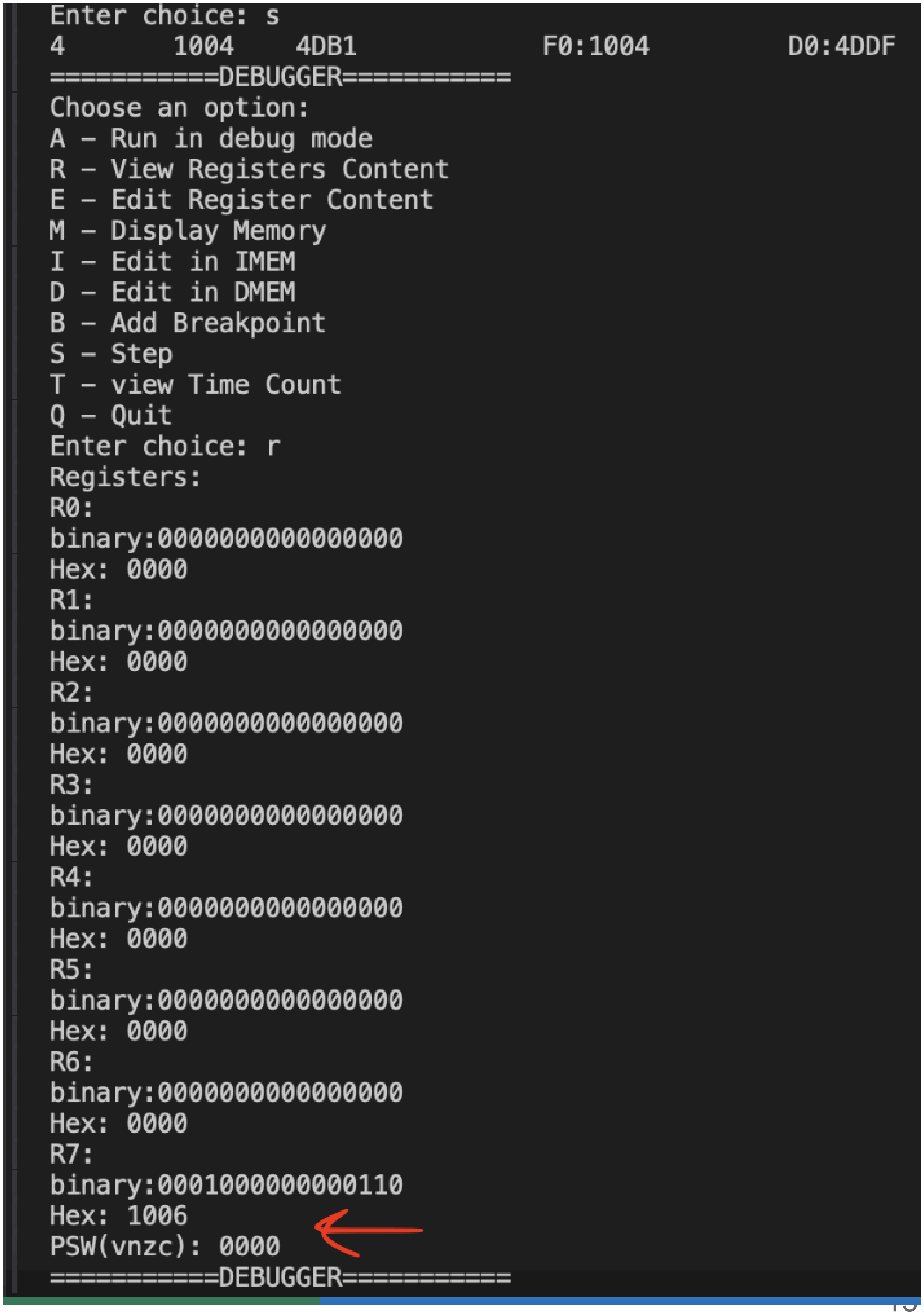
```
  9                     end MAIN
Successful completion of assembly - 1P

** Symbol table **

Constants (Equates)
Name                  Type    Value   Decimal

Labels (Code)
Name                  Type    Value   Decimal
DONE                  REL     1008    4104    PRI
MAIN                  REL     1000    4096    PRI

Labels (Data)
Name                  Type    Value   Decimal

Registers
Name                  Type    Value   Decimal
R7                    REG     0007    7       PRI
R6                    REG     0006    6       PRI
R5                    REG     0005    5       PRI
R4                    REG     0004    4       PRI
R3                    REG     0003    3       PRI
R2                    REG     0002    2       PRI
R1                    REG     0001    1       PRI
R0                    REG     0000    0       PRI

.XME file: \\Mac\Home\Desktop\Computer Architecture\Assembler\scpsw.xme
```



```
@AbdullaSadoun →/workspaces/XM23p (main) $ ./main
===========MENU===========
l - Load file
m - Print memory
f - Fetch (BETA)
d - Debug (BETA)
q - Quit
Enter choice: l
Enter filename: scpsw.xme
scpsw.asmT was loaded succefully
```

Expected Results: The newly implemented CLRCC should work as expected clearing all the fields then only the vc fields in the second call in the PSW register.

Actual Results: The actual result was as expected and the instruction ran as it should, clearing all the fields iin the PSW register.

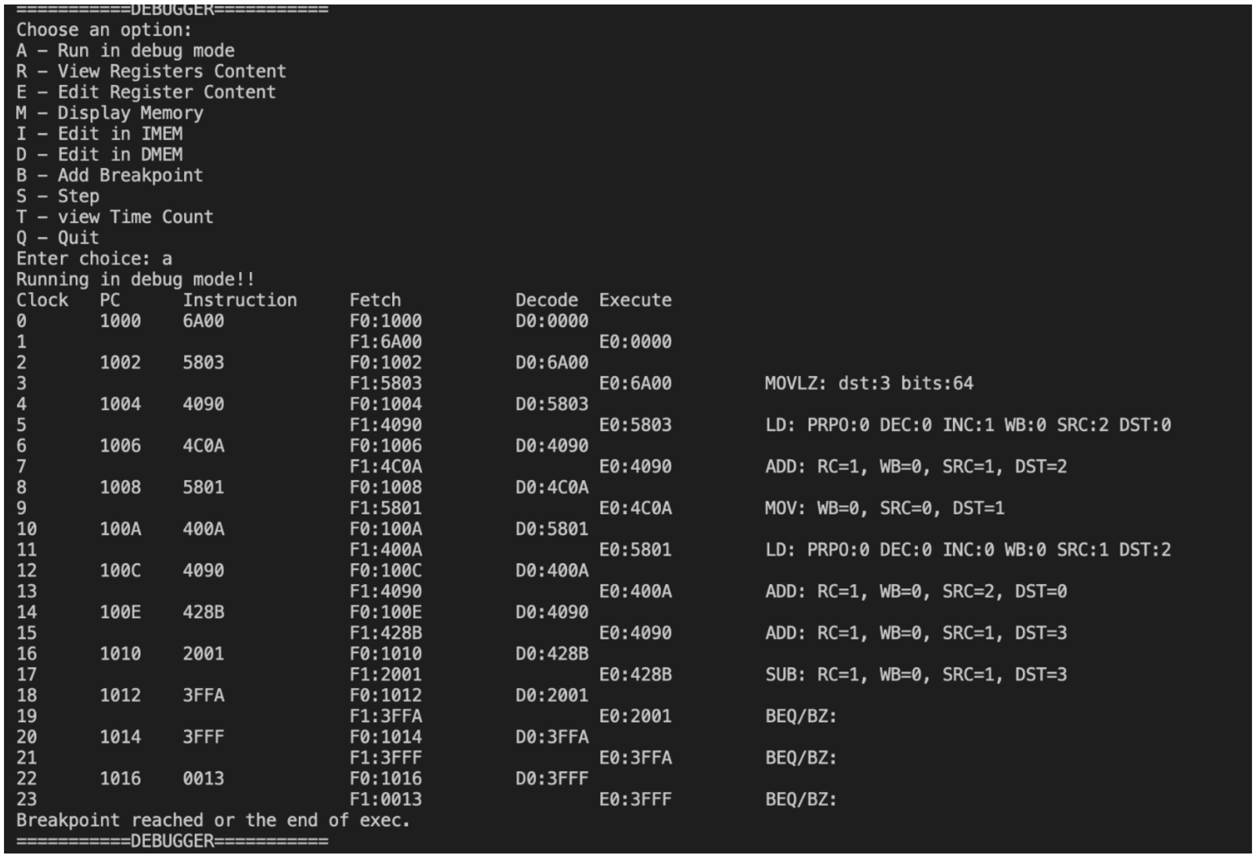


```
Enter choice: s
4        1004     4DB1             F0:1004          D0:4DDF
===========DEBUGGER===========
Choose an option:
A - Run in debug mode
R - View Registers Content
E - Edit Register Content
M - Display Memory
I - Edit in IMEM
D - Edit in DMEM
B - Add Breakpoint
S - Step
T - view Time Count
Q - Quit
Enter choice: r
Registers:
R0:
binary:0000000000000000
Hex: 0000
R1:
binary:0000000000000000
Hex: 0000
R2:
binary:0000000000000000
Hex: 0000
R3:
binary:0000000000000000
Hex: 0000
R4:
binary:0000000000000000
Hex: 0000
R5:
binary:0000000000000000
Hex: 0000
R6:
binary:0000000000000000
Hex: 0000
R7:
binary:0001000000000110
Hex: 1006
PSW(vnzc): 0000
===========DEBUGGER===========
```

First call of CLRCC
Pass/Fail: Pass

Test 3: Running the code in “debugger mode”

Purpose/Objective: The objective of this test is to test the newly implemented execute or run method that goes through the code showing greater detail in the debugger mode like viewing the different stages the CPU emulator is going through when executing the instructions loaded into IMEM.
Test Configuration: For this test, I will try to run the following code acquired from lab1 that has been loaded into IMEM in this new debugger mode. (PART1.xme and PART1.list are included below in Extra Content and Notes)
Expected Results: The program should run through the code in IMEM without running into any issues, it should display the different stages the CPU emulator is in per time tick in CPU clock time.
Actual Results: The program performed as expected successfully running all the instructions in IMEM showing the correct stages for the entire code in IMEM.

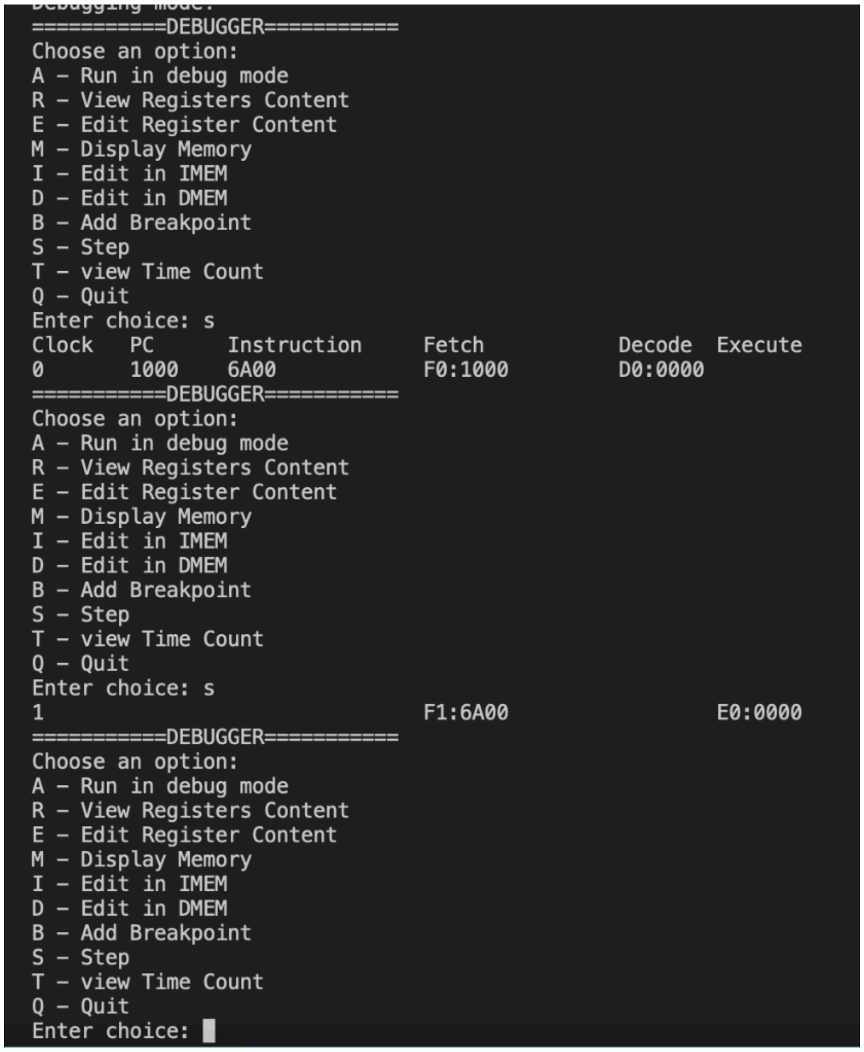


```
===========DEBUGGER===========
Choose an option:
A - Run in debug mode
R - View Registers Content
E - Edit Register Content
M - Display Memory
I - Edit in IMEM
D - Edit in DMEM
B - Add Breakpoint
S - Step
T - view Time Count
Q - Quit
Enter choice: a
Running in debug mode!!
Clock   PC      Instruction     Fetch           Decode  Execute
0       1000    6A00            F0:1000         D0:0000
1                               F1:6A00                 E0:0000
2       1002    5803            F0:1002         D0:6A00
3                               F1:5803                 E0:6A00         MOVLZ: dst:3 bits:64
4       1004    4090            F0:1004         D0:5803
5                               F1:4090                 E0:5803         LD: PRPO:0 DEC:0 INC:1 WB:0 SRC:2 DST:0
6       1006    4C0A            F0:1006         D0:4090
7                               F1:4C0A                 E0:4090         ADD: RC=1, WB=0, SRC=1, DST=2
8       1008    5801            F0:1008         D0:4C0A
9                               F1:5801                 E0:4C0A         MOV: WB=0, SRC=0, DST=1
10      100A    400A            F0:100A         D0:5801
11                              F1:400A                 E0:5801         LD: PRPO:0 DEC:0 INC:0 WB:0 SRC:1 DST:2
12      100C    4090            F0:100C         D0:400A
13                              F1:4090                 E0:400A         ADD: RC=1, WB=0, SRC=2, DST=0
14      100E    428B            F0:100E         D0:4090
15                              F1:428B                 E0:4090         ADD: RC=1, WB=0, SRC=1, DST=3
16      1010    2001            F0:1010         D0:428B
17                              F1:2001                 E0:428B         SUB: RC=1, WB=0, SRC=1, DST=3
18      1012    3FFA            F0:1012         D0:2001
19                              F1:3FFA                 E0:2001         BEQ/BZ:
20      1014    3FFF            F0:1014         D0:3FFA
21                              F1:3FFF                 E0:3FFA         BEQ/BZ:
22      1016    0013            F0:1016         D0:3FFF
23                              F1:0013                 E0:3FFF         BEQ/BZ:
Breakpoint reached or the end of exec.
===========DEBUGGER===========
```

Pass/Fail: Pass

Test 4: Stepping through in “debugger mode”

Purpose/Objective: The objective of this test is to test the newly implemented step through function which can be used to run or execute the code while viewing at fields in greater detail like which stage the CPU emulator is in when stepping through lines of code in the instructions loaded into IMEM.
Test Configuration: For this test, I will try to run the PART1 code acquired from lab1 that has been loaded into IMEM in this new debugger step through mode.
(PART1.xme and PART1.list are included below in Extra Content and Notes)
Expected Results: The program should run through the code in IMEM everytime the s is pressed to step through, it should do so without running into any issues, it should display the different stages the CPU emulator is in per time tick in CPU clock time.
Actual Results: The program performed as expected successfully running all the instructions in IMEM showing the correct stages it is going through everytime it steps through.

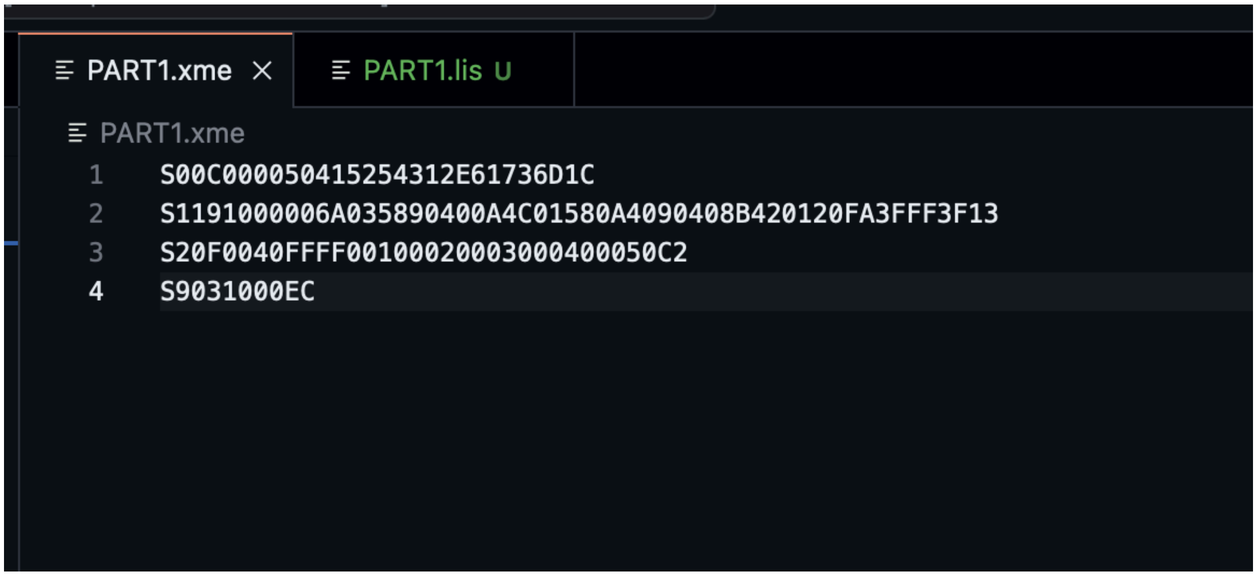


```
===========DEBUGGER===========
Choose an option:
A - Run in debug mode
R - View Registers Content
E - Edit Register Content
M - Display Memory
I - Edit in IMEM
D - Edit in DMEM
B - Add Breakpoint
S - Step
T - view Time Count
Q - Quit
Enter choice: s
Clock   PC      Instruction     Fetch           Decode  Execute
0       1000    6A00            F0:1000         D0:0000
===========DEBUGGER===========
Choose an option:
A - Run in debug mode
R - View Registers Content
E - Edit Register Content
M - Display Memory
I - Edit in IMEM
D - Edit in DMEM
B - Add Breakpoint
S - Step
T - view Time Count
Q - Quit
Enter choice: s
1                               F1:6A00                 E0:0000
===========DEBUGGER===========
Choose an option:
A - Run in debug mode
R - View Registers Content
E - Edit Register Content
M - Display Memory
I - Edit in IMEM
D - Edit in DMEM
B - Add Breakpoint
S - Step
T - view Time Count
Q - Quit
Enter choice: 
```

Pass/Fail: Pass

Extra Content and Notes

PART1.xme (file used)



```
S00C000050415254312E61736D1C
S1191000006A035890400A4C01580A4090408B420120FA3FFF3F13
S20F0040FFFF00100020003000400050C2
S9031000EC
```

PART1.lis (file used)

X-Makina Assembler - Version XM-23P Single Pass+ Assembler - Release 24.04.17
Input file name: PART1.asm
Time of assembly: Wed 15 May 2024 21:33:36

```
 1
 2                          ;
 3                          ; Sum an array of 16-bit numbers
 4                          ; ECED 3403
 5                          ; 15 May 24
 6                          ;
 7                                  CODE
 8                                  org     #1000
 9                          ;
10                          ;
11      1000    6A00    Main    movlz   Array,R0        ; r0=Address of the array
12      1002    5803            ld      R0,R3           ; load stopper into r3
13      1004    4090            add     #2,R0           ; move r0 to the next element (first element
to be summed) increment by 2 as bytes are in pairs
14      1006    4C0A            mov     R1,R2           ; setting r2 as sum register and making it 0
15                          ;
16      1008    5801    loop    ld      R0,R1           ; load the array's element into r1
```

```
100A   400A          add     R1,R2          ; Add the element to the sum
100C   4090          add     #2,R0          ; Increment R0 to point to the next element
in the array
                ;
                ; check if stopper is 0 to stop summing
                ;
100E   428B          sub     #1,R3          ; stopper - 1
1010   2001          bz      Done           ; end loop if stopper is 0
                ;
1012   3FFA          bra     loop           ; continue adding
                ;
                ; adding complete, result are in r2
                ;
                Done
                ;
                ; Finished - busy wait
                ;
1014   3FFF     BWait bra    BWait
                ;
                ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                ;
                ; Data space
                ;
                     DATA
                     org     #40
                ;
                ; the array of integers used:
                ;
0040   FFFF     Array word   $-1            ; (5=stopper in r3)
0042   1000          word    #1000
0044   2000          word    #2000
0046   3000          word    #3000
0048   4000       word       #4000
004A   5000       word       #5000
                ;
                ; no store for result they remained in register
                ;
                     end     Main
Successful completion of assembly - 2P

** Symbol table **

Constants (Equates)
```

```
Name                        Type   Value  Decimal

Labels (Code)
Name                        Type   Value  Decimal
BWait                       REL    1014   4116    PRI
Done                        REL    1014   4116    PRI
loop                        REL    1008   4104    PRI
Main                        REL    1000   4096    PRI

Labels (Data)
Name                        Type   Value  Decimal
Array                       REL    0040   64      PRI

Registers
Name                        Type   Value  Decimal
R7                          REG    0007   7       PRI
R6                          REG    0006   6       PRI
R5                          REG    0005   5       PRI
R4                          REG    0004   4       PRI
R3                          REG    0003   3       PRI
R2                          REG    0002   2       PRI
R1                          REG    0001   1       PRI
R0                          REG    0000   0       PRI

.XME file: \\Mac\Home\Desktop\Computer Architecture\Lab 1\PART1.xme
```

scpsw.lis (file used)

```
X-Makina Assembler - Version XM-23P Single Pass+ Assembler - Release 24.04.17
Input file name: scpsw.asm
Time of assembly: Tue 9 Jul 2024 04:56:08
 1                              org #1000
 2                     MAIN
 3     1000   4DBF            setcc   vsnzc
 4     1002   4DDF            clrcc   vsnzc
 5     1004   4DB1        setcc   vc
 6     1006   4DD1            clrcc   vc
 7                     DONE
 8     1008   3FFF            bra          DONE              ; loop
 9                     end MAIN
Successful completion of assembly - 1P
```

```
** Symbol table **

Constants (Equates)
Name                Type  Value  Decimal

Labels (Code)
Name                Type  Value  Decimal
DONE                REL   1008   4104   PRI
MAIN                REL   1000   4096   PRI

Labels (Data)
Name                Type  Value  Decimal

Registers
Name                Type  Value  Decimal
R7                  REG   0007   7      PRI
R6                  REG   0006   6      PRI
R5                  REG   0005   5      PRI
R4                  REG   0004   4      PRI
R3                  REG   0003   3      PRI
R2                  REG   0002   2      PRI
R1                  REG   0001   1      PRI
R0                  REG   0000   0      PRI

.XME file: \\Mac\Home\Desktop\Computer Architecture\Assembler\scpsw.xme
```

scpsw.xme (file used):

```
S00C000073637073772E61736D54
S10D1000BF4DDF4DB14DD14DFF3F50
S9031000EC
```