

ECED 3403 Computer Architecture Assignment 3 Testing Report

Connor McLeod B00851621

<https://github.com/weakbox/xm23-emu-eced3403>

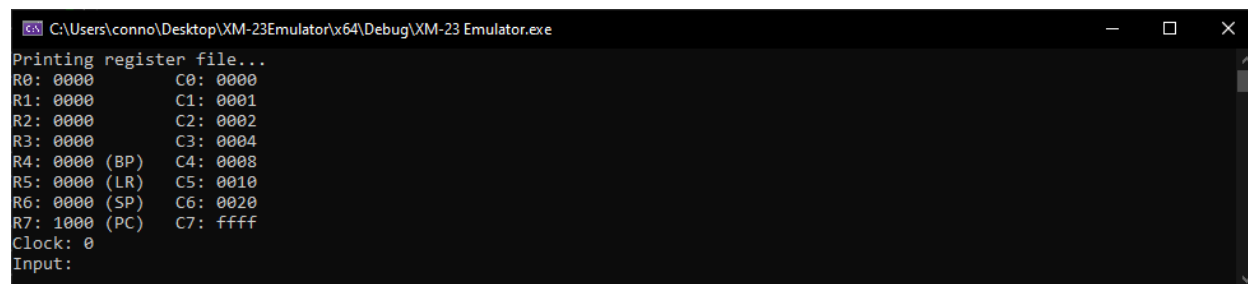
Note: For these tests, the “VERBOSE” flag is defined in the file “header.h”. This flag can be enabled or disabled depending on the level of visual feedback the user would like. Relevant to these tests, the verbose flag prints the fault ID to the console when a fault is encountered.

Test 1: Test Nested Supervisory Calls

Setup: The provided “exceptions_basic.asm” assembly file was assembled using the most recent version of the provided XM-23 assembler. The resulting .xme file was then run using the XM-23 emulator.

Expected Results: We expect to see the emulator branch to and return from the correct address when an SVC instruction is called. This includes nested SVC instructions, which demonstrate the use of the stack.

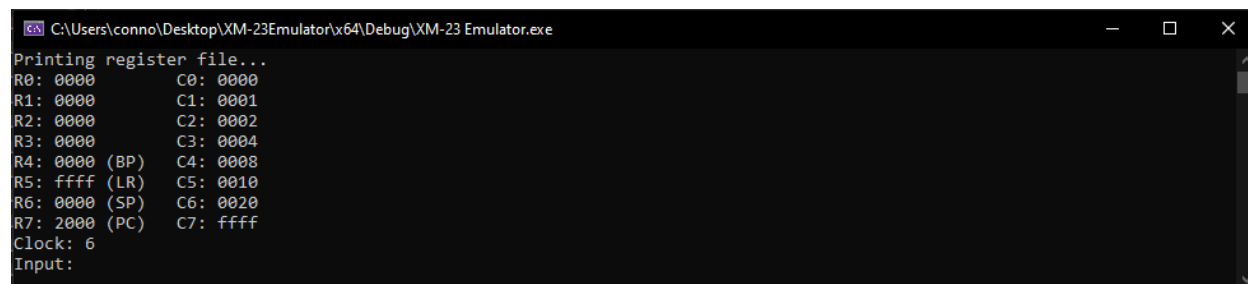
Results: The emulator successfully navigates the program as expected.



```

C:\Users\conno\Desktop\XM-23Emulator\x64\Debug\XM-23 Emulator.exe
Printing register file...
R0: 0000      C0: 0000
R1: 0000      C1: 0001
R2: 0000      C2: 0002
R3: 0000      C3: 0004
R4: 0000 (BP) C4: 0008
R5: 0000 (LR) C5: 0010
R6: 0000 (SP) C6: 0020
R7: 1000 (PC) C7: ffff
Clock: 0
Input:
  
```

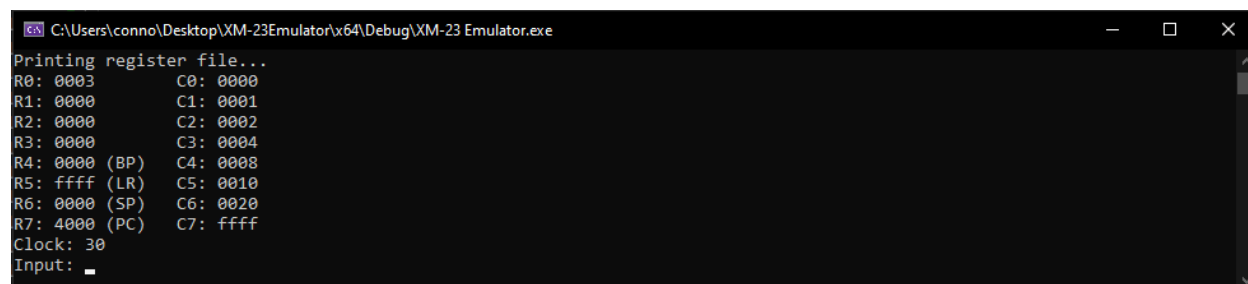
The program counter is initialized to 0x1000.



```

C:\Users\conno\Desktop\XM-23Emulator\x64\Debug\XM-23 Emulator.exe
Printing register file...
R0: 0000      C0: 0000
R1: 0000      C1: 0001
R2: 0000      C2: 0002
R3: 0000      C3: 0004
R4: 0000 (BP) C4: 0008
R5: ffff (LR)  C5: 0010
R6: 0000 (SP) C6: 0020
R7: 2000 (PC) C7: ffff
Clock: 6
Input:
  
```

After the first SVC instruction, we move to the address found in interrupt vector 2, which is 0x2000.



```

C:\Users\conno\Desktop\XM-23Emulator\x64\Debug\XM-23 Emulator.exe
Printing register file...
R0: 0003      C0: 0000
R1: 0000      C1: 0001
R2: 0000      C2: 0002
R3: 0000      C3: 0004
R4: 0000 (BP) C4: 0008
R5: ffff (LR)  C5: 0010
R6: 0000 (SP) C6: 0020
R7: 4000 (PC) C7: ffff
Clock: 30
Input:
  
```

After the second SVC instruction, we move to the address found in interrupt vector 4, which is 0x4000.

```
C:\Users\conno\Desktop\XM-23Emulator\x64\Debug\XM-23 Emulator.exe
Printing register file...
R0: 0003      C0: 0000
R1: 0003      C1: 0001
R2: 0000      C2: 0002
R3: 0000      C3: 0004
R4: 0000 (BP) C4: 0008
R5: ffff (LR) C5: 0010
R6: 0000 (SP) C6: 0020
R7: 2008 (PC) C7: ffff
Clock: 54
Input:
```

After the second SVC instruction is completed, the program counter is restored.

```
C:\Users\conno\Desktop\XM-23Emulator\x64\Debug\XM-23 Emulator.exe
IR: 4c2f (MOV)
Clock: 60
Input: 6
Printing register file...
R0: 0003      C0: 0000
R1: 0003      C1: 0001
R2: 0000      C2: 0002
R3: 0000      C3: 0004
R4: 0000 (BP) C4: 0008
R5: 0000 (LR) C5: 0010
R6: 0000 (SP) C6: 0020
R7: 1002 (PC) C7: ffff
Clock: 60
Input:
```

After the first SVC instruction is completed, the program counter is restored.

```
exceptions_basic.lis - Notepad
File Edit Format View Help
56
57          org #1000
58
59          ; Try to execute an interrupt from the interrupt vector table.
60
61          Start
62
63      1000    4D92    svc $2          ; We should move to address #2000.
64
65      1002    3FFE    bra Start
66
67          ; Create a primitive interrupt handler.
68
69          org #2000
70
71          Interrupt_Handler
72
73      2000    4088    add $1,R0
74      2002    4088    add $1,R0
75      2004    4088    add $1,R0
76      2006    4D94    svc $4          ; We should move to address #4000.
77      2008    4C2F    mov R5,R7      ; Move LR into PC (interrupt return).
78
79          org #4000
80
81      4000    4089    add $1,R1
82      4002    4089    add $1,R1
83      4004    4089    add $1,R1
84      4006    4C2F    mov R5,R7      ; Move LR into PC (interrupt return).
85
86          end Start
```

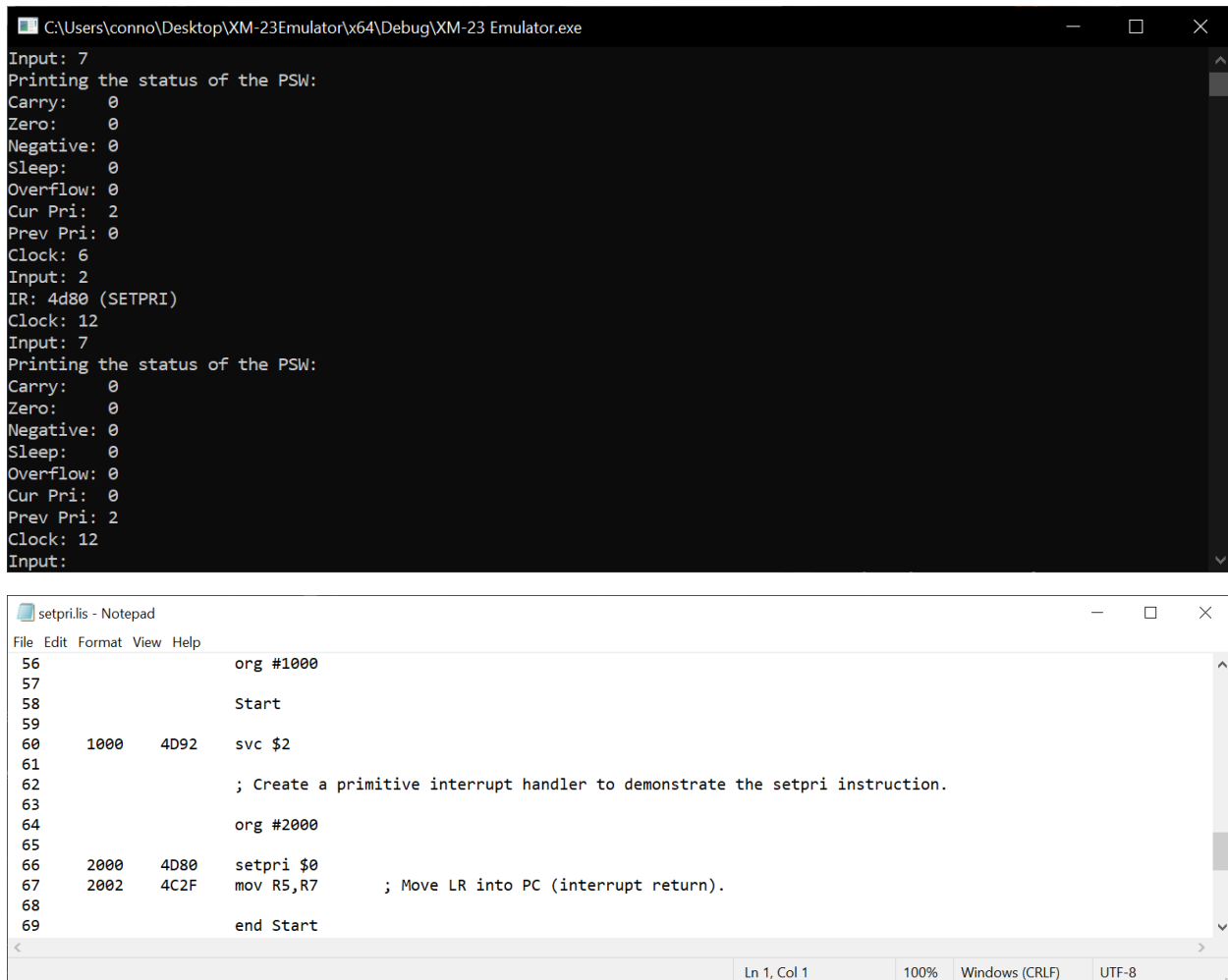
Pass/Fail: Pass. The emulator performs as expected.

Test 2: Test SETPRI Instruction

Setup: The provided “setpri.asm” assembly file was assembled using the most recent version of the provided XM-23 assembler. The resulting .xme file was then run using the XM-23 emulator.

Expected Results: We expect to see that after the program has trapped to an interrupt vector with priority 2, we can change that interrupts priority with the SETPRI instruction. In this case, we will change it to 0.

Results: The emulator correctly modifies the priority of the interrupt.



The top screenshot shows the XM-23 Emulator window. The title bar is "C:\Users\conno\Desktop\XM-23Emulator\x64\Debug\XM-23 Emulator.exe". The output shows the program's state before and after an interrupt. Before the interrupt, the current priority (Cur Pri) is 2. After the interrupt (IR: 4d80 (SETPRI)), the current priority is updated to 0. The status of the Program Status Word (PSW) is printed twice, showing the change in priority.

The bottom screenshot shows the assembly code in a Notepad window titled "setpri.lis - Notepad". The code defines a primitive interrupt handler. It starts at address 1000, sets up a service instruction (svc \$2) at address 1000, and then uses the SETPRI instruction at address 2000 to set the priority to 0. The code ends at address 2002 with a move instruction (mov R5, R7) and an end statement.

```

56                                     org #1000
57
58                                     Start
59
60      1000      4D92      svc $2
61
62                                     ; Create a primitive interrupt handler to demonstrate the setpri instruction.
63
64                                     org #2000
65
66      2000      4D80      setpri $0
67      2002      4C2F      mov R5,R7      ; Move LR into PC (interrupt return).
68
69                                     end Start
  
```

Pass/Fail: Pass. The emulator performs as expected.

Test 3: Test Illegal Instruction Fault

Setup: The provided “fault_ill_inst.asm” assembly file was assembled using the most recent version of the provided XM-23 assembler. The resulting .xme file was then run using the XM-23 emulator.

Expected Results: We expect to see that the emulator correctly identifies the illegal instruction and calls the correct fault handler. The fault handler will put a value of 0xFFFF into R0, indicating that an illegal instruction fault has occurred. While this isn’t a very robust fault handler, it at least tells us that a programmer could theoretically create a much more feature-packed fault handler in their own code.

Results: The emulator correctly identifies the illegal instruction and executes the proper fault handler (8).

```

C:\Users\conno\Desktop\XM-23Emulator\Debug\XM-23 Emulator.exe
Input: 2
IR: 4da0
Instruction not found!
Fault triggered!
ID: 8
Clock: 18
Input: 6
Printing register file...
R0: 0000      C0: 0000
R1: 0000      C1: 0001
R2: 0000      C2: 0002
R3: 0000      C3: 0004
R4: 0000 (BP) C4: 0008
R5: ffff (LR) C5: 0010
R6: 0000 (SP) C6: 0020
R7: 8000 (PC) C7: ffff
Clock: 18
Input: 2
IR: 4288 (SUB)
Clock: 24
Input: 6
Printing register file...
R0: ffff      C0: 0000
R1: 0000      C1: 0001
R2: 0000      C2: 0002
R3: 0000      C3: 0004
R4: 0000 (BP) C4: 0008
R5: ffff (LR) C5: 0010
R6: 0000 (SP) C6: 0020
R7: 8002 (PC) C7: ffff
Clock: 24
Input:

```

```

fault_ill_inst.lis - Notepad
File Edit Format View Help
56
57          org #1004
58
59      1004      4DA0      word #4DA0          ; Put an illegal instruction at address 1004.
60
61          org #1000
62
63      Start
64
65      1000      4080      add $0,R0
66      1002      4080      add $0,R0
67                          ; Bad instruction.
68
69                          ; Create a primitive interrupt handler to document an illegal instruction.
70
71          org #8000
72
73      Interrupt_Handler
74
75      8000      4288      sub $1,R0          ; When we encounter the illegal instruction we should see #FFFF in R0.
76      8002      4C2F      mov R5,R7          ; Move LR into PC (interrupt return).
77
78          end Start

```

Pass/Fail: Pass. The emulator performs as expected.

Test 4: Test Invalid Address Fault

Setup: The provided “fault_inv_addr.asm” assembly file was assembled using the most recent version of the provided XM-23 assembler. The resulting .xme file was then run using the XM-23 emulator.

Expected Results: We expect to see that the emulator correctly identifies the invalid address while fetching the program counter and calls the correct fault handler. The fault handler will put a value of 0xFFFF into R0, indicating that an illegal instruction fault has occurred.

Results: The emulator correctly identifies the invalid address and executes the proper fault handler (9).

```

C:\Users\conno\Desktop\XM-23Emulator\x64\Debug\XM-23 Emulator.exe
Input: 6
Printing register file...
R0: 0000      C0: 0000
R1: 1001      C1: 0001
R2: 0000      C2: 0002
R3: 0000      C3: 0004
R4: 0000 (BP) C4: 0008
R5: 0000 (LR) C5: 0010
R6: 0000 (SP) C6: 0020
R7: 1001 (PC) C7: ffff
Clock: 18
Input: 2
Fault triggered!
ID: 9
IR: 4288 (SUB)
Clock: 24
Input: 6
Printing register file...
R0: ffff      C0: 0000
R1: 1001      C1: 0001
R2: 0000      C2: 0002
R3: 0000      C3: 0004
R4: 0000 (BP) C4: 0008
R5: ffff (LR) C5: 0010
R6: 0000 (SP) C6: 0020
R7: 9002 (PC) C7: ffff
Clock: 24
Input:

```

```

fault_inv_addr.lis - Notepad
File Edit Format View Help
58          org #1000
59
60          Start
61
62      1000    6009    movl BAD_ADDR,R1
63      1002    7881    movh BAD_ADDR,R1
64      1004    4C0F    mov R1,R7          ; Move an odd-numbered address into the program counter.
65
66          ; Create a primitive interrupt handler to document an illegal instruction.
67
68          org #9000
69
70          Interrupt_Handler
71
72      9000    4288    sub $1,R0          ; When we encounter the invalid address we should see #FFFF in R0.
73      9002    4C2F    mov R5,R7          ; Move LR into PC (interrupt return).
74
75          end Start

```

Pass/Fail: Pass. The emulator performs as expected.

Test 5: Test Priority Fault

Setup: The provided “fault_pri.asm” assembly file was assembled using the most recent version of the provided XM-23 assembler. The resulting .xme file was then run using the XM-23 emulator.

Expected Results: We expect to see that the emulator correctly identifies the priority faults while executing the SETPRI and SVC instructions and calls the correct fault handler. The fault handler will put a value of 0xFFFF into R0, indicating that an illegal instruction fault has occurred.

Results: The emulator correctly identifies the priority faults and executes the proper fault handler (10).

```

C:\Users\conno\Desktop\XM-23Emulator\x64\Debug\XM-23 Emulator.exe
Input: 2
IR: 4d87 (SETPRI)
Fault triggered!
ID: 10
Clock: 6
Input: 2
IR: 4288 (SUB)
Clock: 12
Input: 2
IR: 4c2f (MOV)
Clock: 18
Input: 6
Printing register file...
R0: ffff      C0: 0000
R1: 0000      C1: 0001
R2: 0000      C2: 0002
R3: 0000      C3: 0004
R4: 0000 (BP) C4: 0008
R5: 0000 (LR) C5: 0010
R6: 0000 (SP) C6: 0020
R7: 1002 (PC) C7: ffff
Clock: 18
Input: 2
IR: 4d96 (SVC)
Clock: 24
Input: 2
IR: 4d92 (SVC)
Fault triggered!
ID: 10
Clock: 30
Input: 2
IR: 4288 (SUB)
Clock: 36
Input: 2
IR: 4c2f (MOV)
Clock: 42
Input: 6
Printing register file...
R0: fffe      C0: 0000
R1: 0000      C1: 0001
R2: 0000      C2: 0002
R3: 0000      C3: 0004
R4: 0000 (BP) C4: 0008
R5: ffff (LR) C5: 0010
R6: 0000 (SP) C6: 0020
R7: 2000 (PC) C7: ffff
Clock: 42
Input:

```

```

fault_pri.lis - Notepad
File Edit Format View Help
56          org #1000
57
58          Start
59
60      1000    4D87    setpri $7
61      1002    4D96    svc $6
62
63          ; Create a primitive interrupt handler to document a priority fault.
64
65          org #2000
66
67          ;      Nothing here!
68
69          org #6000
70
71      6000    4D92    svc $2 ; This will trigger a priority fault as we are trying to trap to a process with lower priority.
72
73          org #A000
74
75          Interrupt_Handler
76
77      A000    4288    sub $1,R0      ; When we encounter the priority fault we should see #FFFF in R0.
78      A002    4C2F    mov R5,R7      ; Move LR into PC (interrupt return).
79
80          end Start

```

Ln 1, Col 1 100% Windows (CRLF) UTF-8

Pass/Fail: Pass. The emulator performs as expected.