Dalhousie University
Department of Electrical and Computer Engineering

ECED 3403 – Computer Architecture

Lab 4: Changing condition codes and Displaying stages

1 Objectives

This lab consists of two parts.

First, to add the instructions SETCC (set condition codes) and CLRCC (clear condition codes) to your emulator. This can help make testing easier and put a program into a known state.

Second, to display the clock and the stages as a program runs in debug mode. This can also make testing easier as it shows what is occurring in each stage at each clock step.

2 Changing condition codes

In this part, the SETCC and CLRCC instructions are to be designed, implemented, and tested as part of your emulator.

The instructions SETCC and CLRCC instructions do not work on registers, they set or clear the four condition code bits (V, N, Z, C) and the SLP bit. The SLP bit puts the CPU to sleep until a higher-priority exception (an interrupt) occurs; the SLP bit can be ignored.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Mnemonic	Instruction
0	1	0	0	1	1	0	1	1	0	1	V	SLP	N	Z	C	SETCC	Set PSW bits (1 = set)
0	1	0	0	1	1	0	1	1	1	0	V	SLP	N	Z	C	CLRCC	Clear PSW bits (1 = clear)

If a bit is to be set or cleared, the corresponding bit in the instruction is set. If the bit is not to be changed, it has a value of zero; for example, to set the V and Z bits, SETCC would have V=1, Z=1, SLP=0, N=0, and C=0. If the C bit is to be cleared, CLRCC would have C=1 and the remaining operand bits would be zero.

Only the bits that are set in the operand cause a change to the PSW bits. Bits in the operand that are not set (i.e., zero) do not change the current value of the PSW bits.

One or more condition codes can be set or cleared using the assembly instructions:

```
      setcc       C     ; PSW.C <- 1
      clrcc       C     ; PSW.C <- 0
;
      setcc       VNZC  ; Set all CC bits
      clrcc       CZNV  ; Clear all CC bits
```

The order of bits being set or cleared is immaterial, although spaces are not permitted. The assembler creates the instruction specifying the bits to be changed.

2.1 Examples

The SETCC and CLRCC instructions can be used for a variety of applications.

Returning a Boolean value from a function

A subroutine might be designed to return a TRUE or FALSE indication:

```
IF Check_System(X, Y, Z) THEN
      PRINT “System Functioning”
ELSE
      PRINT “System Error”
ENDIF
```

The Boolean value from `Check_System()` could be returned in a register (a possible waste of a register) or by changing the value of the Carry-bit to indicate `TRUE` (`PSW.C←1`) or `FALSE` (`PSW.C←0`):

```
; TRUE
      SETCC C
      MOV   LR,PC
...
; FALSE
      CLRCC C
      MOV   LR,PC
```

Initializing a value

The RRC instruction moves a register's LSbit through the Carry bit and the current value of the Carry bit to the register's MSBit. If we use RRC without knowing the original value of the Carry bit, the program could enter an unknown state. By setting or clearing the Carry bit before using RRC, we can be sure of the program's state:

```
;
; Count number of bits set in a register
;
      movlz #0,R1       ; R1 <- 0 (bit count)
;
; Bit count loop
;
Loop  cmp   #0,R0       ; When word-with-bits is 0, stop
      beq   Done
;
      clrcc c           ; Carry <- 0
;
      rrc   R0          ; R0.bit15 <- C; R0 <- R0 >> 1; C <- RRC.bit0
;
      addc  #0,R1       ; R1 <- R1 + Carry (0|1)
;
      bra   Loop
```

CLRCC or SETCC can be used to put a program into a known state before using the DADD instruction, which includes the carry when adding the first pair of nibbles.

3 Displaying stages

In this part of the lab you are to extend your debugger so that when a program runs in debug mode, it displays the clock (in decimal), the program counter, the state of the stages at the current clock value. For example (the instruction can be displayed as either its mnemonic or the 16-bit hex value):

Clock	PC	Instruction	Fetch	Decode	Execute
0	2000	400A	F0: 2000	D0: 4C00	
1			F1: 400A		E0: 4C00
2	2002	4C10	F0: 2002	D0: 400A	
3			F1: 4C10		E0: 400A

The solution is to be designed, implemented, tested, and demonstrated.

If you already have this feature in your A2, submit the relevant code to the Lab 4 submission page.

Your solution can be added to your A2 software. Only submit those parts of A2 that have been changed.

3.1 Example

As an example, consider the following S-Record file that is read into the machine's memories (you will need to demonstrate your own S-Record files):

```
S00B00004C6162322E61736D44
S10B20000A40104CA24782497A
S9032000DC
```

When your program is to run the loaded program, it should begin at the starting address specified in the S-Record file.

Clock	PC	Instruction	Fetch	Decode	Execute
0	2000	400A	F0: 2000	D0: 4C00	
1			F1: 400A		E0: 4C00
2	2002	4C10	F0: 2002	D0: 400A	
3			F1: 4C10		E0: 400A
4	2004	47A2	F0: 2004	D0: 4C10	
5			F1: 47A2		E0: 4C10
6	2006	4982	F0: 2006	D0: 47A2	
7			F1: 4982		E0: 47A2

Your solution is to display the stages being performed at each clock tick, both even and odd.

3.2 Suggestions (not requirements)

Breakpoints could be extended to stop on a specific clock cycle rather than an address. Similarly, a version of single step could step through the code using time rather than the address of the next instruction.

4 Submission and Marking

4.1 Submission

Each part will be marked using the following marking scheme:

Design: A brief design is required, consisting of a data dictionary of the major structures and algorithms for decoding and storing.

Software: A fully commented, indented, magic-numberless, tidy piece of software that meets the requirements described above and follows the design description.

Testing: Sufficient test results must be supplied to demonstrate that your implementation meets the specifications.

You should give a demonstration of your solution during the lab or after class.

Your design, software, and tests must be submitted electronically to the course website.

4.2 Marking

Marking is as follows:

3: All requirements (design, software, and implementation) met.

2: One requirement missing or coding problem.

1: Two or three requirements missing or coding problems.

0: Not submitted or more than three requirements missing or coding problems.

Undemonstrated solutions will not be marked.

5 Important Dates

Available: 3 July 2024 at noon.

Submission due: 5 July 2024 at 11:59pm, Atlantic.

6 Miscellaneous

This lab is to be completed individually.

If you are having *any* difficulty with this or any part of the course, *please* contact Dr. Hughes or Emad as soon as possible.