Dalhousie University
Department of Electrical and Computer Engineering

ECED 3403 – Computer Architecture

Assignment 5: Conditional execution

Objectives

Short conditional instructions such as:

```
IF Data = 8 THEN
      Alpha <- 1;
ELSE
      Alpha <- 2;
ENDIF
```

Produces several labels and branches:

```
;
; Data (R0), Alpha (R1)
;
; IF Data = 8 THEN
      cmp         $8,R0
      bne         ElsePart
; Alpha <- 1;
      movlz       $1,R1
      bra         EndIf
; ELSE
ElsePart
; Alpha <- 2;
      movlz       $2,R1
;
; ENDIF
;
EndIF
```

Some machines allow instructions other than branches to be executed conditionally. XM23 has this capability with its conditional execution, or CEX, instruction. In this, the final assignment in ECED 3403, you are to implement the CEX instruction.

Background

Branching in a RISC can be expensive. Some architectures, like ARM's 32-bit ISA allow instructions to be executed conditionally. Each instruction has a four-bit condition prefix (the four most significant bits) that indicates the conditions under which it will execute. The execution codes and their meaning are summarized in Table 1.

Table 1: ARM's conditional execution condition prefixes and meanings

Assembler code	Description	PSW bit values inspected	Condition Prefix
`EQ`	Equal / equals zero	Z = 1	`0000`
`NE`	Not equal	Z = 0	`0001`
`CS / HS`	Carry set / unsigned higher or same	C = 1	`0010`
`CC / LO`	Carry clear / unsigned lower	C = 0	`0011`
`MI`	Minus / negative	N = 1	`0100`
`PL`	Plus / positive or zero	N = 0	`0101`
`VS`	Overflow	V = 1	`0110`
`VC`	No overflow	V = 0	`0111`
`HI`	Unsigned higher	C = 1 and Z = 0	`1000`
`LS`	Unsigned lower or same	C = 0 or Z = 1	`1001`
`GE`	Signed greater than or equal	N = V	`1010`
`LT`	Signed less than	N ≠ V	`1011`
`GT`	Signed greater than	Z = 0 and (N = V)	`1100`
`LE`	Signed less than or equal	Z = 1 or (N ≠ V)	`1101`
`TR`	True part is always executed	Ignored	`1110`
`FL`	False part is always executed	Ignored	`1111`

This allows instructions to be executed conditionally (by default, they are always executed), for example:

- There is only one branch instruction (B), which can be turned into a conditional:

```
B        Loop  ; Unconditional branch
BVS      Loop  ; Branch if oVerflow set
BLE      Loop  ; Branch if less-than or equal (Z=1 or N ≠ V)
```

- Instructions such as ADD and SUB can execute conditionally:

```
ADDEQ    R1,R2 ; IF PSW.Z=1 THEN R2 <- R2 + R1 ELSE Bubble ENDIF
SUBMI    R0,R8 ; IF PSW.N=1 THEN R8 <- R8 - R0 ELSE Bubble ENDIF
```

ARM's 16-bit ISA is a compressed version of the 32-bit ISA (Thumb mode) has, amongst other things, the conditional prefix removed. However, conditional execution is still supported using the IT "instruction" which informs the ARM 16-to-32-bit frontend to convert the incoming 16-bit instructions to be converted into 32-bit instructions with a condition (all other instructions are converted to 32-bits without the condition codes being set).

The format of the IT (If-Then) instruction is (the Conditions are listed in Table 1):

```
IT + 0 {[T|F]} 3 + Condition
```

Up to four instructions can be executed conditionally; the first instruction (the first 'T' following the 'I') is always true and always executed; for example:

```
ITTFF    EQ    ; Execute first 2 if EQ or second 2 if NE
ADD      #1,R4 ; Add #1 to R4 if EQ
ADD      #2,R3 ; Add #2 to R3 if EQ
SUB      #1,R4 ; Sub #1 from R4 if NE
MOV      R3,R5 ; Move R3 to R5 if NE
```

The instructions are still fetched; however, the fetch stage determines whether the instruction should be executed or bubbled.

The CEX instruction

XM23 has a Conditional EXecution instruction, CEX, it allows up to eight instructions to be executed, depending on the condition. The instruction format is as follows:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	0	0	C	C	C	C	T	T	T	F	F	F

When the Fetch stage detects a CEX instruction (in step F1), it determines which PSW bits are to be tested by inspecting the code bits, C, bits 6 through 9). (The different codes, the suffix used by the assembler to determine the code, the description of the code, and the PSW bits to be tested are shown in Table 1, above.) Bits 3 through 5 indicate the number of instructions to be executed if the true-state condition is met (T, true-count) and bits 0 through 2 is the count of instructions to be executed if the condition is not met (F, false-count). The t-count and f-count values need not be equal, and the order of the conditional execution code bits is unrelated to the order of the PSW bits.

The code is organized into the conditional instruction, CEX, followed by the instructions to be executed if the condition is TRUE and the number of instructions to be executed if the condition is FALSE. If the condition is TRUE, any FALSE instructions must be fetched and bubbled; if the condition is FALSE, the TRUE instruction must be fetched and bubbled, and the FALSE instructions fetched and executed:

```
CEX instruction
0 to t-count instructions to be executed if TRUE, bubbled otherwise
0 to f-count instructions to be executed if FALSE, bubbled otherwise
```

The instruction is written as:

```
      CEX   <COND>,<T-COUNT>,<F-COUNT>
```

The assembler produces the correct instruction based on the condition and counts.

Example

The following pseudo-code:

```
IF Data = 8 THEN
      Alpha <- 1;
ELSE
      Alpha <- 2;
ENDIF
```

This can be implemented as:

```
;
; Assume Data, Alpha, and Beta are R0, R1, and R2, respectively
; IF Data = 8 THEN
      cmp   $8,R0
;
; Test the PSW bit for the EQ condition (Z=1)
; If Z=1, then do the instruction following the CEX, otherwise bubble
;
      cex        eq,$1,$1
```

```
;
     movlz     $1,R1 ; Executed if EQ, bubbled if NE
;
     movlz     $2,R1 ; Bubbled if EQ, executed if NE
;
```

The CEX instruction means that the number of conditional branches increases. For example, to branch if the oVerflow bit is set, we could write:

```
     CEX   VS,1,0
     BRA   Overflow_Found
```

Note the f-count of ‘`0`’ is required to indicate that the overflow bit is clear because control does not pass to `Overflow_Found`.

The CEX state information is kept as part of the Fetch stage.

Marking

The assignment will be marked using the following marking scheme:

Design: The design description must include a brief introduction as to the problem being solved (0.5), a design section (3), and a data dictionary (1.5).

Total points: 5.

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

Total points: 11.

Testing: In addition to any test software supplied as part of the assignment, you will be responsible for developing sufficient distinct tests demonstrating that the software operates according to the design specification. Some of the tests should exercise the software. The tests must include the name of the test, its purpose or objective, the test configuration, and the test results.

Total points: 4.

All three sections are to be submitted electronically.

The loader must be demonstrated before the software and testing will be marked.

Important Dates

Available: 16 July 2024

Design document due: 30 July 2024 at 11:59pm, Atlantic.

Software and testing due: 7 August 2024 at 11:59pm, Atlantic.

Demonstration: 6 and 7 August 2024. Your implementation **must** be demonstrated before the software and testing will be marked.

Late submissions will not be accepted after 7 August 2024 at 11:59pm.

Miscellaneous

This assignment is to be completed individually.

The work is to be implemented in C. It is recommended that you use Visual Studio for this assignment.

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

This assignment is worth 5% of your overall course grade.

This is a non-trivial assignment. It should be started as soon as it is made available.

Assignments that are found to be copies of work done by other students or has used AI will result in an immediate dismissal from this course.