COMP2120 Assignment 4

April 27, 2025

1. Hand assemble the following assembly code and put it in a program file. Run the simulator on this program. Explain what the function `SQ` does?

```
    SUB     R4,R4,R4        0000H: 01040404
    LD      P1,R1           0004H: 0600ff01 00000078
    MOV     R1,R2           000CH: 05010002
    LD      P2,R3           0010H: 0600ff03 0000007c
L:  MOV     R1,R10          0018H: 0501000a
    CALL    SQ              001CH: 0c00ff00 00000044
    ADD     R4,R11,R4       0024H: 00040b04
    ADD     R1,R2,R1        0028H: 00010201
    SUB     R3,R1,R5        002CH: 01030105
    BNZ     L               0030H: 0802ff00 00000018
    ST      R4,P            0038H: 0704ff00 00000080
    HLT                     0040H: 09000000

/* Procedure to calculate ____, input is R10, output is R11 */
/* The proc uses R12 and R13, need to save them on entry */
/* and restore them when exit*/

SQ: PUSH    R12             0044H: ....
    PUSH    R13             0048H: ....
    LD      P1,R13          004CH: ....
    SUB     R11,R11,R11     0054H:
    MOV     R10,R12         0058H:
L2: ADD     R11,R10,R11     005CH:
    SUB     R12,R13,R12     0060H:
    BNZ     L2              0064H:
    POP     R13             006CH:
    POP     R12             0070H:
    RET                     0074H:
P1: .WORD   1               0078H: 00000001
P2: .WORD   A               007CH: 0000000a
P:  .WORD                   0080H: 00000000
```

Solution: The function `SQ` calculates the square of `R10` and stores the result in `R11`.
In other words, `R11` = `R10`× `R10`.
The following is the hand assembled code:

```
0044H:  0a0c0000
0048H:  0a0d0000
004CH:  0600ff0d 00000078
0054H:  010b0b0b
0058H:  050a000c
005CH:  000b0a0b
0060H:  010c0d0c
0064H:  0802ff00 0000005c
006CH:  0b00000d
0070H:  0b00000c
0074H:  0d000000
```

You can also find it in the attached file `prog1`

2. Run the simulator in debug mode. Write down the data transfer/transformation sequences involved in the execution of the instructions `CALL` and `RET`. You may skip intermediate step provided by the simulator, for example the instruction fetching step should look like:

   ```
   MAR <- PC
   IR  <- mem[MAR]
   ```

   or in English, move the value of `PC` to `MAR`. Then read memory and the result(`mem[MAR]`) is moved to `IR`, i.e. just write down the source and destination of the data movement, without the paths etc.

Solution: data transfer/transformation sequences:
In `CALL`:

```
MAR <- PC
MBR <- mem[MAR]
MAR <- MBR
PC <- PC + 4
TEMP <- MAR
SP <- SP - 4
MAR <- SP
MBR <- PC
mem[MAR] <- MBR
MAR <- TEMP
PC <- MAR
```

In `RET`:

```
MAR <- SP
SP <- SP + 4
MBR <- mem[MAR]
PC <- MBR
```

3. Modify the program so that it will calculate the value of $1-2+3-4\cdots-8+9$. That is,

```
sum = 0;
for i = 1 to 9 do sum += sq(i)
```

Where `sq(i)` return `i` when `i` is odd, otherwise return `-i`. Note that the original program is already a loop from 1 to 9. Just replace the function `SQ` by

```
if (R10 is odd) R11 = R10;
else R11 = 0 - R10;
```

Since we don't have a `NEG` instruction, to find $-x$, we use $0-x$. To check if a number x is odd, just check if the rightmost is 1. We can find x `AND` $00000000\cdots0001$. (i.e. 1) After `AND` operation, all bits `AND`ed with 0 will be 0. If the rightmost bit is 0, then the result is 0. Otherwise the result is non-zero. Note that the address of `P1`, `P2` and `P` may got changed when the length of the function is changed. You may need to change the address of them in the program, e.g. in line 2

```
LD  P1,R1
```

you may need to find the new address of `P1`, and also in line 4 …

Solution: The following is the modified program and corresponding assembled code:

```
    SUB     R4,R4,R4    0000H:  01040404
    LD      P1,R1       0004H:  0600ff01 0000007c
    MOV     R1,R2       000CH:  05010002
    LD      P2,R3       0010H:  0600ff03 00000080
L:  MOV     R1,R10      0018H:  0501000a
    CALL    SQ          001CH:  0c00ff00 00000044
    ADD     R4,R11,R4   0024H:  00040b04
    ADD     R1,R2,R1    0028H:  00010201
    SUB     R3,R1,R5    002CH:  01030105
    BNZ     L           0030H:  0802ff00 00000018
    ST      R4,P        0038H:  0704ff00 00000084
    HLT                 0040H:  09000000
SQ: PUSH    R12         0044H:  0a0c0000
    PUSH    R13         0048H:  0a0d0000
    LD      P1,R13      004CH:  0600ff0d 0000007c
    SUB     R11,R11,R11 0054H:  010b0b0b
    MOV     R10,R12     0058H:  050a000c
    AND     R12,R13,R13 005CH:  030c0d0d
    BNZ     L2          0060H:  0802ff00 0000006c
    SUB     R11,R12,R12 0068H:  010b0c0c
L2: MOV     R12,R11     006CH:  050c000b
    POP     R13         0070H:  0b00000d
    POP     R12         0074H:  0b00000c
    RET                 0078H:  0d000000
P1: .WORD   1           007CH:  00000001
P2: .WORD   A           0080H:  0000000a
P:  .WORD               0084H:  00000000
```

You can also find it in the attached file `prog3`