ISE1/EE2 Computer Architecture
Problem Sheets 1 to 6 with Solutions

Problem Sheet 1

1. Convert 512_{10} to a 16-bit two's complement binary number [P&H, Ex.4.1]
2. Convert $-1,023_{10}$ to a 16-bit two's complement binary number [P&H, Ex.4.2]
3. What hexadecimal number does the binary number 0110111_2 represent? What decimal number does it represent?
4. What decimal number does the unsigned binary number 11.01 represent? What would it represent as a signed (two's complement) binary number?
5. If a memory has a 16-bit address bus and a 32-bit data bus, what is the largest size the memory could be?
6. What does the ASCII hex sequence 41 52 4D represent?
7. What is the largest positive number that can be represented in (integer) two's complement using n bits? The largest negative number?

Problem Sheet 1: ANSWERS

1. 0000 0010 0000 0000 (NOT 1111 1110 0000 0000)
2. 1111 1100 0000 0001 (NOT 0000 0011 1111 1111)
3. 37, 55
4. 3.25, -0.75
5. 256 kbytes
6. ARM
7. $2^{n-1} - 1$, -2^{n-1}

Problem Sheet 2

1. Consider the Pascal/Delphi statement

   ```
   f := (a + b) - (c + d);
   ```

   Translate this statement into MU0 assembly language.

2. Consider the Pascal/Delphi statement

   ```
   if (a = b) f := a+b else f := a-b;
   ```

   Translate this statement into MU0 assembly language.

3. How many clock cycles would your code for the above statements take to complete on MU0?

Problem Sheet 2: ANSWERS

1.
```
LDA  c   ; 2 cycles
ADD  d   ; 2 cycles
STO  t   ; 2 cycles
LDA  a   ; 2 cycles
ADD  b   ; 2 cycles
SUB  t   ; 2 cycles
STO  f   ; 2 cycles
STP      ; 1 cycle, total = 15 cycles
```

2.
```
    LDA  a    ; 2 cycles
    SUB  b    ; 2 cycles
    JNE  L1   ; 1 cycle
    LDA  a    ; 2 cycles
    ADD  b    ; 2 cycles
    JMP  L2   ; 1 cycles
L1: LDA  a    ; 2 cycles
    SUB  b    ; 2 cycles
L2: STP       ; 1 cycle, total = 11 (‘if branch’) or 10 ('else branch’)
```
   Other solutions are possible.

3. Exec times for instructions are in notes and are shown above.

Problem Sheet 3

1. MU0 uses 1-address instructions. Re-write the code from Problem Sheet 2, Question 1 for

   (a) A 3-address instruction format machine with instruction set below.

   ```
   ADD m1 m2 m3;   mem[m3] = mem[m1] + mem[m2]  (Takes 4 cycles)
   SUB m1 m2 m3;   mem[m3] = mem[m1] - mem[m2]  (Takes 4 cycles)
   STP         ;   stop execution               (Takes 1 cycle)
   ```

   (b) A 2-address instruction format machine with instruction set below.

   ```
   ADD m1 m2;   mem[m2] = mem[m1] + mem[m2]   (Takes 4 cycles)
   SUB m1 m2;   mem[m2] = mem[m1] - mem[m2]   (Takes 4 cycles)
   STP      ;   stop execution                (Takes 1  cycle)
   ```

2. Assume that the 3-address machine has a 4-bit opcode and three 12-bit addresses for each instruction. Assume that the 2-address machine has an 8-bit opcode and two 12-bit addresses for each instruction. Which machine (including MU0) has the smallest code? Which machine runs the code fastest?

3. Do you think these results are generalizable to other algorithms apart from Problem Sheet 2, Question 1?

Problem Sheet 3: ANSWERS

1. (a) 3-address machine:

```
ADD a b x;  5 bytes, 4 cycles
ADD c d y;  5 bytes, 4 cycles
SUB x y f;  5 bytes, 4 cycles
STP      ;  5 bytes, 1 cycle
```

Totals: 20 bytes, 13 cycles.

(b) 2-address machine:

```
ADD a b;   4 bytes, 4 cycles
ADD c d;   4 bytes, 4 cycles
SUB b d;   4 bytes, 4 cycles
STP    ;   4 bytes, 1 cycle
```

Totals: 16 bytes, 13 cycles. (Note that this solution destroys the input data - preserving it would take more instructions).

(c) (MU0)

```
LDA  c   ; 2 bytes, 2 cycles
ADD  d   ; 2 bytes, 2 cycles
STO  t   ; 2 bytes, 2 cycles
LDA  a   ; 2 bytes, 2 cycles
ADD  b   ; 2 bytes, 2 cycles
SUB  t   ; 2 bytes, 2 cycles
STO  f   ; 2 bytes, 2 cycles
STP      ; 2 bytes, 1 cycle
```

Totals: 16 bytes, 15 cycles.

2. Smallest code: Both MU0 and the 2-address machine have the smallest code.

   Fastest code: Both the 3-address machine and the 2-address machine have the fastest code.

3. This question is intended to provoke discussion.

Problem Sheet 4

1. Consider the following ARM program [Furber, p.75]. This program calls a subroutine `HexOut` to convert the number `VALUE` into hexadecimal for screen display. `SWI 0x11` exits the program and `SWI 0x00` writes the character in the bottom 8-bits of `r0` to the display.

```
             AREA  Hex_Out, CODE,READONLY
SWI_WriteC   EQU   &0
SWI_Exit     EQU   &11
             ENTRY
             LDR   r1, VALUE
             BL    HexOut
             SWI   SWI_Exit
VALUE        DCD   &12345678
HexOut       MOV   r2, #8
LOOP         MOV   r0, r1, LSR #28
             CMP   r0, #9
             ADDGT r0, r0, #'A'-10
             ADDLE r0, r0, #'0'
             SWI   SWI_WriteC
             MOV   r1, r1, LSL #4
             SUBS  r2, r2, #1
             BNE   LOOP
             MOV   pc, r14
             END
```

   (a) Which registers are affected by the `BL` instruction?

   (b) What does the instruction `MOV r0, r1, LSR #28` achieve?

   (c) What is the difference between the value `#0` and `#'0'`? Why are the values `#'A'-10` or `#'0'` added to `r0`?

   (d) What is the effect of the instruction `MOV r1, r1, LSL #4`?

2. Re-write your answers to Problem Sheet 2, Questions 1 and 2, this time in ARM code.

3. F_i, the ith factorial, is defined as below.

$$F_i = \begin{cases} i \cdot F_{i-1}, & i \geq 1 \\ 1, & i = 0 \end{cases} \tag{1}$$

Some ARM code to calculate the $i = 3$rd factorial follows.

```
        AREA  Fact_Seq, CODE, READONLY

        ENTRY
        LDR   r0, i               ;  1
        BL    Fact                ;  2
        STR   r1, f               ;  3
        SWI   &11                 ;  4
i       DCD   &3                  ;  5
f       DCD   &0                  ;  6

; Subroutine Fact
; Input: r0 contains desired factorial
; Output: r1 contains the result
Fact    STMED r13!, {r0,r2,r14}   ;  7
        CMP   r0, #0              ;  8
        MOVEQ r1, #1              ;  9
        BEQ   FactRet             ; 10
        MOV   r2, r0              ; 11
        SUB   r0, r0, #1          ; 12
        BL    Fact                ; 13
        MUL   r1, r2, r1          ; 14
FactRet LDMED r13!, {r0,r2,r14}   ; 15
        MOV   pc, r14             ; 16

        END
```

How many BL instructions are executed? Illustrate the stack contents just after each STMED instruction has completed.

Problem Sheet 4: ANSWERS

1. (a) The program counter (`PC`) and the link register (`r14` or `lr`).

   (b) It moves the top 4-bits of `r1` into `r0`.

   (c) `#0` is the value `0`, `#'0'` is the value `0x30`, corresponding to the ASCII code of '0'. The values `#'A'-10` (for digits `0xA` to `0xF`) or `#'0'` (for digits `0x0` to `0x9`) are added to convert from the number in `r0` to its equivalent ASCII code required by `SWI 0x00`.

   (d) This instruction shifts `r1` left by four bits so that the next nybble to display is the most significant nybble.

2. (a)
```
LDR r0, a
LDR r1, b
ADD r2, r0, r1
LDR r0, c
LDR r1, d
ADD r0, r0, r1
SUB r0, r3, r0
STR r0, f
```

   (b)
```
LDR   r0, a
LDR   r1, b
SUBS  r2, r0, r1
ADDEQ r2, r0, r1
STR   r2, f
```

   Other solutions will probably crop up here, but this solution has the nice use of `SUBS` both as a subtraction and a comparison, and the use of `ADDEQ` to override the 'else' result with the 'if' case.

3. BL's are executed in the following line number order: 2 (non-recursive), 13 (depth-1), 13 (depth-2), 13 (depth-3). Total = 4.

   After 1st STMED: <ret-addr1>, ?, 3
   After 2nd STMED: <ret-addr1>, ?, 3, <ret-addr2>, 3, 2
   After 3rd STMED: <ret-addr1>, ?, 3, <ret-addr2>, 3, 2, <ret-addr2>, 2, 1
   After 4th STMED: <ret-addr1>, ?, 3, <ret-addr2>, 3, 2, <ret-addr2>, 2, 1, <ret-addr2>, 1, 0

   Here <ret-addr1> is the address of Line 3 and <ret-addr2> is the address of Line 14.

Problem Sheet 5

Assemble the following ARM code by hand. State the contents of each address location (in binary or hex). You may assume that execution starts from address 0x0.

Hint: You may wish to use the program counter as a base register for loads and stores. Note that due to the pipelining on the ARM, the program counter is 8 bytes ahead of the currently executing load/store instruction.

```
          AREA Example, CODE
          ENTRY

          LDR   r0, a
          LDR   r1, b
          CMP   r0, r1
          BNE   label1
          ADD   r0, r1, r0
label1    STR   r0, f
          SWI   0x11
a         DCD   0x40
b         DCD   0x50
f         DCD   0xFF

          END
```

Problem Sheet 5: ANSWERS

Address	Contents	Assembly
0x00	1110 01 0 1 1 0 0 1 1111 0000 0000 0001 0100	LDR r0, a
0x04	1110 01 0 1 1 0 0 1 1111 0001 0000 0001 0100	LDR r0, b
0x08	1110 00 0 1010 1 0000 XXXX 00000 XX 0 0001	CMP r0, r1
0x0C	0001 101 0 0000 0000 0000 0000 0000 0000	BNE label1
0x10	1110 00 0 0100 0 0001 0000 00000 XX 0 0000	ADD r0, r1, r0
0x14	1110 01 0 1 1 0 0 0 1111 0000 0000 0000 1000	STR r0, f
0x18	1110 1111 000000000000000000010001	SWI 0x11
0x1C	0000 0000 0000 0000 0000 0000 0100 0000	DCD 0x40
0x20	0000 0000 0000 0000 0000 0000 0101 0000	DCD 0x50
0x24	0000 0000 0000 0000 0000 0000 1111 1111	DCD 0xFF

Problem Sheet 6

Consider the code fragment shown below.

```
for i := 1 to 10
    A[i] := i;

for i := 1 to 5
    A[i] := A[2*i-1] + A[2*i];
```

1. Assume that array A consists of 10 consecutive blocks in memory, starting at location 0x0 (i.e. A[1] has memory location 0x0). Construct a time-ordered list of block numbers which are accessed by the code fragment, and indicate which is a read and which is a write.
2. If each of the above memory accesses takes 100ns, what is the time taken for the execution of all memory accesses in this code fragment?
3. We decide to introduce a direct-mapped cache with 8 blocks. Which of the above accesses corresponds to a 'hit', and which to a 'miss'?
4. If each hit takes 10ns and each miss takes 100ns, what is the overall time taken now that the data is cached? (you can assume a write-back cache)

Problem Sheet 6: ANSWERS

1.
   ```
   0x0 (write)
   0x1 (write)
   0x2 (write)
   0x3 (write)
   0x4 (write)
   0x5 (write)
   0x6 (write)
   0x7 (write)
   0x8 (write)
   0x9 (write)

   0x0 (read)
   0x1 (read)
   0x0 (write)
   0x2 (read)
   0x3 (read)
   0x1 (write)
   0x4 (read)
   0x5 (read)
   0x2 (write)
   0x6 (read)
   0x7 (read)
   0x3 (write)
   0x8 (read)
   0x9 (read)
   0x4 (write)
   ```

   (There will be variations in these answers depending on whether `A[2*i-1]` or `A[2*i]` was deemed to be read first)

2. $100 * (10 + 3 * 5) = 2500ns = 2.5\mu s$

3.
   ```
   0x0 (miss, block 0)
   0x1 (miss, block 1)
   0x2 (miss, block 2)
   0x3 (miss, block 3)
   0x4 (miss, block 4)
   0x5 (miss, block 5)
   0x6 (miss, block 6)
   0x7 (miss, block 7)
   0x8 (miss, block 0)
   0x9 (miss, block 1)

   0x0 (miss, block 0)
   0x1 (miss, block 1)
   ```

```
0x0 (hit, block 2)
0x2 (hit, block 2)
0x3 (hit, block 3)
0x1 (hit, block 1)
0x4 (hit, block 4)
0x5 (hit, block 5)
0x2 (hit, block 2)
0x6 (hit, block 6)
0x7 (hit, block 7)
0x3 (hit, block 3)
0x8 (miss, block 0)
0x9 (miss, block 1)
0x4 (hit, block 4)
```

4. $14 * 100 + 11 * 10 = 1510ns = 1.51\mu s$