Homework Lecture 5

4.1

A memory has 2^{24} addressable locations. What is the smallest width in bits that the address can be while still being able to address all 2^{24} locations?

4.5

If a branch is taken in the ARC processor the destination address is:
%PC+(4×sign_extension(simm22))

a) What is the range of the branch target relative to the PC in bytes?
b) What is the range in words?

4.7

a) sethi 0xABCD, %r12
b) call label_d
c) orcc %r15, 255, %r22
d) be label_d
e) st %r25, [%r9 + 128]
f) srl %r8, 31, %r9

label_d is 64 bytes ahead of the instruction in which it is referenced.
Encode the previous ARC instructions in Binary and Hex.

Note

1. 4.7b should be **call label_d** (error in the book)
2. The call instruction uses a relative address. The explanation on page 120 first gives the impression that it is an absolute address ("call a subroutine that begins at location sub_r") and in the next sentience "sub_r is 25 words (100 bytes) further in memory. The latter is correct. You can verify this using the ARCTools.

4.8

a) sethi 0xABCDEF, %r12
b) call 0xFFFB
c) or %r15, 0x1FFF, %r22
d) be -4
e) st %r25, [%r9+128]
f) srl %r8, 32, %r9

Which of the previous instructions are legal, and if not, why not?
Verify your answer using the assembler of the ARCtools

4.12

Write an ARC program that performs a swap operation on the 32-bit operands x=25 and y=50, which are stored in memory. Use as few registers as you can.
Simulate with the ARCtools

4.13,

Note: the syntax of the bold lines is changed (compared with the assignment in the book)
First determine the behavior yourself and then check it by means of a simulation with the ARCtools.

```
        .begin
        .org 0
Y:      ld      [k], %r1
        addcc   %r1, -4, %r1
        st      %r1, [k]
        bneg    X
        ld      [%r1 + a], %r2
        ld      [%r1 + b], %r3
        addcc   %r2, %r3, %r4
        st      %r4, [%r1 + c]
        ba      Y
X:      halt
k:      40
a:      1,2,3,4,5,6,7,8,9,10
b:      1,2,3,4,5,6,7,8,9,10
c:
        .end
```

4.18

A program compiled for a SPARC ISA writes the 32-bit unsigned integer 0xABCDEF01 to a file and reads it back correctly. The same program compiled for a Pentium ISA also works correctly. However, when the file is transferred between machines the program incorrectly reads the integer from the file as 0x01EFCDAB. What is going wrong?