Reading 4

Brockman 3.2.4 Data Transfer Instructions

- albaCore has two data transfer instructions:
  - ↳ load (ld)
  - ↳ store (st)
- Note load immediate (ldi) is also used

- which command in albaCore copies data from memory to a register?
  - ↳ "ld"

- Suppose r1 contains 0x1230 and r2 contains 0xabc0. What does the following instruction do?

  st r1, r2, 4

  ↳ Write 0x1230 to memory location 0xabc4 a.k.a. M[r2+4] ← r1

- What is the machine code for the following instruction?

  st r1, r2, 4

  ↳ 0x9412

  ↳ NOTE: The field order is op code, offset, source register, base address register.

- Consider the following fragment from an albaCore assembly language program

  .data
  A: 0x1234

What is the value of high(A)?

↳ Can't tell from info provided. high(A) is the high byte of the address of A. It has nothing to do with the data stored at A.

3.2.5 Branch Instructions

- Branch instructions specify the address after the branch is taken in terms of a displacement from current pc value. This displacement can be positive or negative.

- Branch instructions can either be unconditional or conditional.
  - ↳ for conditional: if condition is met then program execution continues with instructions at memory address pc + displacement. Otherwise it continues at the next sequential address.

- albaCore's 3 main branch instructions:
  1. branch unconditionally (br)
  2. branch if register rb equals 0 (bz)
  3. branch if register rb is negative (bn)

- Branch instructions are used for if statements and loops
- Branch instructions use relative displacement, especially for conditionals and loops.
- Branch instruction 0xa800 branches backwards since the displacement is 0x80 which is negative since the MSB is 1.
- When a label is used as an argument to a branch instruction, the assembler calculates the displacement from the current address to the address of the label and substitutes the displacement into the instructions.

3.2.6 Jump Instructions

- jump instructions change the program counter to some new absolute address.
  - ↳ primarily used to implement procedure calls and returns.

- jump instruction (jr) changes program counter to 16-bit value contained in source register ra.

- jump and link instruction (jal) provides the mechanism for calling procedures and knowing where to return from them. It does 2 things:
  1. it changes program counter to an absolute target location that is specified with the instruction
  2. it saves the value of the old program counter plus one in register 15.

- Instead of specifying the argument to jal as a numerical target address, you can instead specify the label of the starting instruction of the function.

- What is written to r15 when the following instruction is executed?

  jal Oxabc

  ↳ the current value of the program counter + 1, then it jumps to Oxabc for the next instruction

- True or False: The jr instruction always returns execution to the instruction after a jal instruction.

  ↳ False, only if the register argument, typically r15, contains that address.

3.3 Writing Longer Assembly Programs

Original C code

```
int x = 1;
int y = 2;
int z;

if (x < y)
   z = 1;
else
   z = 2;
x = x + z;
```

Using goto statements

```
   int tmp;
   int x;
   int y;
   int z;

   tmp = x - y;
   if (tmp < 0) goto if_block;
else_block:
   z = 2;
   goto cont;
if_block:
   z = 1;
cont:
   x = x + z;
```

Assembly

```
                        //     int tmp;   // r0
  ldi r1, 1             //     int x = 1; // r1
  ldi r2, 2             //     int y = 2; // r2
                        //     int z;     // r3
                        //
  sub r0, r1, r2        //     tmp = x - y;
  bn  r0, if_block      //     if (tmp < 0) goto if_block;
else_block:             // else_block:
  ldi r3, 2             //     z = 2;
  br  cont              //     goto cont:
if_block:               // if_block:
  ldi r3, 1             //     z = 1;
cont:                   // cont:
  add r1, r1, r3        //     x = x + z
```

Another C → goto → Assembly example

C

```c
#include <stdio.h>
int A[8] = {3, 1, 4, 2, 15, 5, 32, 9};

int main()
{
    int max = 0;
    int i = 0;
    while (i < 8) {
        if (A[i] > max)
            max = A[i];
        i = i + 1;
    }
    printf("%d\n", max);
}
```

goto

```c
#include <stdio.h>
int A[8] = {3, 1, 4, 2, 15, 5, 32, 9};

int main()
{
    int tmp;
    int max = 0;
    int i = 0;
    int size = 8;
    int d;        // emulates register with data read from memory
    int *a = A;   // emulates writing the starting address of A in a register

while_condition:
    tmp = size - i;
    if (tmp == 0) goto done;
    d = a[i];
    tmp = d - max;
    if (tmp < 0) goto update_i;
    max = d;
update_i:
    i = i + 1;
    goto while_condition;
done:
    printf("%d\n", max);
}
```

Assembly

```
    .data
                          // int A[8] = {3, 1, 4, 2, 15, 5, 32, 9};
A: 3, 1, 4, 2, 15, 5, 32, 9

    .text
main:                     // int main()
                          // {
                          //     int tmp;          // r0
    ldi r1, 0             //     int max = 0;      // r1
    ldi r2, 0             //     int i = 0;        // r2
    ldi r3, 8             //     int size = 8;     // r3
                          //     int d;            // r4
    ldi r5, high(A)       //     int *a = A        // r5
    ldi r0, 8
    shl r5, r5, r0
    ldi r0, low(A)
    or  r5, r5, r0
                          //
while_condition:          // while_condition:
    sub r0, r3, r2        //     tmp = size - i;
    bz  r0, done          //     if (tmp == 0) goto done;
    add r0, r5, r2        //     d = a[i];
    ld  r4, r0, 0
    sub r0, r4, r1        //     tmp = d - max;
    bn  r0, update_i      //     if (tmp < 0) goto update_i;
    or  r1, r4, r4        //     max = d;
update_i:                 // update_i:
    ldi r0, 1             //     i = i + 1;
    add r2, r2, r0
    br  while_condition   //     goto while_condition;
done:                     // done:
    or  r0, r1, r1        //     printf("%d\n", max);
    quit                  // }
```