Computer Organization

Assignment 1

2018-14245 김익환

May 13, 2020

```c
void bubble_sort_iter(int arr[], int n) {
  int temp;
  // 1st for loop
  for (int i = 0; i < n - 1; ++i)
    // 2nd for loop
    for (int j = 0; j < n - i - 1; ++j)
      if (arr[j] > arr[j + 1]) {
        temp = arr[j];
        arr[j] = arr[j + 1];
        arr[j + 1] = temp;
      }
  return;
}
```

Listing 1: Iterative bubble sort in C

```
00010074 <bubble_sort_iter>:
// allocate 12 words in stack
10074:  fd010113        addi    sp,sp,-48
// save s0 at 44(sp), callee saved
10078:  02812623        sw      s0,44(sp)
// use s0 as a frame pointer
1007c:  03010413        addi    s0,sp,48
// store 1st argument, arr[], at -36(s0)
10080:  fca42e23        sw      a0,-36(s0)
// store 2nd argument, n, at -40(s0)
10084:  fcb42c23        sw      a1,-40(s0)
// initialise i with 0 at -20(s0)
10088:  fe042623        sw      zero,-20(s0)
// jumps to the condition statement of 1st for loop
```

```
1008c:  0cc0006f        j       10158 <bubble_sort_iter+0xe4>
// initialise j with 0 at -24(s0)
10090:  fe042423        sw      zero,-24(s0)
// jumps to the condition statement of 2nd for loop
10094:  0a00006f        j       10134 <bubble_sort_iter+0xc0>
// if (arr[j] > arr[j + 1])
10098:  fe842783        lw      a5,-24(s0)
1009c:  00279793        slli    a5,a5,0x2
100a0:  fdc42703        lw      a4,-36(s0)
100a4:  00f707b3        add     a5,a4,a5
100a8:  0007a703        lw      a4,0(a5)
100ac:  fe842783        lw      a5,-24(s0)
100b0:  00178793        addi    a5,a5,1
100b4:  00279793        slli    a5,a5,0x2
100b8:  fdc42683        lw      a3,-36(s0)
100bc:  00f687b3        add     a5,a3,a5
100c0:  0007a783        lw      a5,0(a5)
100c4:  06e7d263        bge     a5,a4,10128 <bubble_sort_iter+0xb4>
// swap arr[j] and arr[j + 1], temp at -28(s0)
100c8:  fe842783        lw      a5,-24(s0)
100cc:  00279793        slli    a5,a5,0x2
100d0:  fdc42703        lw      a4,-36(s0)
100d4:  00f707b3        add     a5,a4,a5
100d8:  0007a783        lw      a5,0(a5)
100dc:  fef42223        sw      a5,-28(s0)
100e0:  fe842783        lw      a5,-24(s0)
100e4:  00178793        addi    a5,a5,1
100e8:  00279793        slli    a5,a5,0x2
100ec:  fdc42703        lw      a4,-36(s0)
100f0:  00f70733        add     a4,a4,a5
100f4:  fe842783        lw      a5,-24(s0)
100f8:  00279793        slli    a5,a5,0x2
100fc:  fdc42683        lw      a3,-36(s0)
10100:  00f687b3        add     a5,a3,a5
10104:  00072703        lw      a4,0(a4)
10108:  00e7a023        sw      a4,0(a5)
1010c:  fe842783        lw      a5,-24(s0)
10110:  00178793        addi    a5,a5,1
10114:  00279793        slli    a5,a5,0x2
10118:  fdc42703        lw      a4,-36(s0)
1011c:  00f707b3        add     a5,a4,a5
10120:  fe442703        lw      a4,-28(s0)
```

```
10124:   00e7a023          sw      a4,0(a5)
// increment j
10128:   fe842783          lw      a5,-24(s0)
1012c:   00178793          addi    a5,a5,1
10130:   fef42423          sw      a5,-24(s0)
// condition statement of 2nd for loop
10134:   fd842703          lw      a4,-40(s0)
10138:   fec42783          lw      a5,-20(s0)
1013c:   40f707b3          sub     a5,a4,a5
10140:   fff78793          addi    a5,a5,-1
10144:   fe842703          lw      a4,-24(s0)
10148:   f4f748e3          blt     a4,a5,10098 <bubble_sort_iter+0x24>
// increment i
1014c:   fec42783          lw      a5,-20(s0)
10150:   00178793          addi    a5,a5,1
10154:   fef42623          sw      a5,-20(s0)
// condition statement of 1st for loop
10158:   fd842783          lw      a5,-40(s0)
1015c:   fff78793          addi    a5,a5,-1
10160:   fec42703          lw      a4,-20(s0)
10164:   f2f746e3          blt     a4,a5,10090 <bubble_sort_iter+0x1c>
// return statement
10168:   00000013          nop
// restore s0 at 44(sp)
1016c:   02c12403          lw      s0,44(sp)
// restore stack pointer
10170:   03010113          addi    sp,sp,48
// return back to the caller function
10174:   00008067          ret
```

Listing 2: Iterative bubble sort in RV32I

```
void bubble_sort_recur(int arr[], int n) {
  if (n == 1) return;
  int temp;
  for (int i = 0; i < n - 1; ++i)
    if (arr[i] > arr[i + 1]) {
      temp = arr[i];
      arr[i] = arr[i + 1];
      arr[i + 1] = temp;
    }
  bubble_sort_recur(arr, n - 1);
  return;
```

```
12 }
```

Listing 3: Recursive bubble sort in C

```
00010074 <bubble_sort_recur>:
// allocate 12 words in stack
10074:  fd010113        addi    sp,sp,-48
// store return address at 44(sp)
10078:  02112623        sw      ra,44(sp)
// store s0 at 40(sp), callee saved
1007c:  02812423        sw      s0,40(sp)
// use s0 as a frame pointer
10080:  03010413        addi    s0,sp,48
// store 1st argument, arr[], at -36(s0)
10084:  fca42e23        sw      a0,-36(s0)
// store 2nd argument, n, at -40(s0)
10088:  fcb42c23        sw      a1,-40(s0)
// if(n == 1)
1008c:  fd842703        lw      a4,-40(s0)
10090:  00100793        li      a5,1
10094:  0cf70a63        beq     a4,a5,10168 <bubble_sort_recur+0xf4>
// initialise i with 0 at -20(s0)
10098:  fe042623        sw      zero,-20(s0)
// jumps to the condition statement of for loop
1009c:  0a00006f        j       1013c <bubble_sort_recur+0xc8>
// if (arr[i]) > arr[i + 1])
100a0:  fec42783        lw      a5,-20(s0)
100a4:  00279793        slli    a5,a5,0x2
100a8:  fdc42703        lw      a4,-36(s0)
100ac:  00f707b3        add     a5,a4,a5
100b0:  0007a703        lw      a4,0(a5)
100b4:  fec42783        lw      a5,-20(s0)
100b8:  00178793        addi    a5,a5,1
100bc:  00279793        slli    a5,a5,0x2
100c0:  fdc42683        lw      a3,-36(s0)
100c4:  00f687b3        add     a5,a3,a5
100c8:  0007a783        lw      a5,0(a5)
100cc:  06e7d263        bge     a5,a4,10130 <bubble_sort_recur+0xbc>
// swap arr[i] and arr[i + 1], temp at -24(s0)
100d0:  fec42783        lw      a5,-20(s0)
100d4:  00279793        slli    a5,a5,0x2
100d8:  fdc42703        lw      a4,-36(s0)
100dc:  00f707b3        add     a5,a4,a5
```

```
100e0:   0007a783          lw      a5,0(a5)
100e4:   fef42423          sw      a5,-24(s0)
100e8:   fec42783          lw      a5,-20(s0)
100ec:   00178793          addi    a5,a5,1
100f0:   00279793          slli    a5,a5,0x2
100f4:   fdc42703          lw      a4,-36(s0)
100f8:   00f70733          add     a4,a4,a5
100fc:   fec42783          lw      a5,-20(s0)
10100:   00279793          slli    a5,a5,0x2
10104:   fdc42683          lw      a3,-36(s0)
10108:   00f687b3          add     a5,a3,a5
1010c:   00072703          lw      a4,0(a4)
10110:   00e7a023          sw      a4,0(a5)
10114:   fec42783          lw      a5,-20(s0)
10118:   00178793          addi    a5,a5,1
1011c:   00279793          slli    a5,a5,0x2
10120:   fdc42703          lw      a4,-36(s0)
10124:   00f707b3          add     a5,a4,a5
10128:   fe842703          lw      a4,-24(s0)
1012c:   00e7a023          sw      a4,0(a5)
// increment i
10130:   fec42783          lw      a5,-20(s0)
10134:   00178793          addi    a5,a5,1
10138:   fef42623          sw      a5,-20(s0)
// condition statement of for loop
1013c:   fd842783          lw      a5,-40(s0)
10140:   fff78793          addi    a5,a5,-1
10144:   fec42703          lw      a4,-20(s0)
10148:   f4f74ce3          blt     a4,a5,100a0 <bubble_sort_recur+0x2c>
// recursive call of bubble_sort_recur
1014c:   fd842783          lw      a5,-40(s0)
10150:   fff78793          addi    a5,a5,-1
10154:   00078593          mv      a1,a5
10158:   fdc42503          lw      a0,-36(s0)
1015c:   f19ff0ef          jal     ra,10074 <bubble_sort_recur>
10160:   00000013          nop
10164:   0080006f          j       1016c <bubble_sort_recur+0xf8>
// return statement
10168:   00000013          nop
// restore return address at 44(sp)
1016c:   02c12083          lw      ra,44(sp)
// restore s0 at 40(sp)
```

```
10170:   02812403          lw      s0,40(sp)
// restore stack pointer
10174:   03010113          addi    sp,sp,48
// return back to the caller function
10178:   00008067          ret
```

Listing 4: Recursive bubble sort in RV32I

1. How are argumnets of `bubble_sort_iter()` and `bubble_sort_recur()` maintained in the stack?

   `bubble_sort_iter`의 경우 12 words(`addi sp, sp, -48`)를 stack에 할당하였다. 그 중 `44(sp)`에 `s0`(callee-saved register)를 저장하였다. 그 후 `s0`에 `48(sp)`를 저장하여 frame pointer로 활용하였다. `bubble_sort_iter`의 첫번째 argument인 `arr[]`는 `-36(s0)`에 저장하였다. 두번째 argument인 `n`은 `-40(s0)`에 저장하였다.

   `bubble_sort_recur`의 경우 12 words(`addi sp, sp, -48`)를 stack에 할당하였다. 그 중 `44(sp)`에 return address를 저장하였고(recursive call을 하므로) `40(sp)`에 `s0`(callee-saved register)를 저장하였다. 그 후 `s0`에 `48(sp)`를 저장하여 frame pointer로 활용하였다. `bubble_sort_recur`의 첫번째 argument인 `arr[]`는 `-36(s0)`에 저장하였다. 두번째 argument인 `n`은 `-40(s0)`에 저장하였다.

2. Does `bubble_sort_iter()` and `bubble_sort_recur()` use `jal`, `jalr`, or both?

   `bubble_sort_iter()`와 `bubble_sort_recur()` 둘 다 `jal`과 `jalr`을 모두 사용하였다. 두 함수 모두 `j`라는 operation과 `ret`라는 operation을 사용하였다. `j`는 unconditional jump를 뜻하는 RV32I의 pseudo-instruction이다. 즉, `j`는 `jal x0, imm`과 같다. 두 함수 모두 `j`를 사용하였으므로 `jal`을 사용하였다. `ret`는 return address로 unconditional jump를 하는 또 다른 pseudo-instruction이다. `ret`는 `jalr x0, 0(x1)`과 같다. 두 함수 모두 `ret`를 사용하였으므로 `jalr`을 사용하였다.

3. How does `bubble_sort_iter()` and `bubble_sort_recur()` restore the stack before returning to a caller function?

   각 RV32I 코드에서 return statement 주석 부분을 보면 된다. `bubble_sort_iter()`는 `s0`에 미리 저장해두었던 `44(sp)` 값을 load하여 `s0`를 복원하였다. `addi sp, sp, 48`을 하여 stack pointer를 복원하였다. `bubble_sort_recur()`는 `x1`에 미리 저장해두었던 `44(sp)` 값을 load하여 return address를 복원하였다. 또한 `s0`에 `40(sp)` 값을 load하여 `s0`를 복원하였다. 그 후 `addi sp, sp, 48`을 하여 stack pointer를 복원하였다. 각 값을 미리 저장해두는 과정은 함수 코드 초기에 있다.

4. What is `ret` instruction shown in `objdump`? (RISC-V ISA does not have `ret` instruction)

`ret`는 return address로 unconditional jump를 명하는 RV32I의 pseudo-instruction 이다. `ret`의 정의는 `jalr x0, 0(x1)`이다. 즉, jump instruction 기준 `pc + 4` 값을 `x0`에 dump하고 return address로 jump한다.

5. Shortly explain how `emu-r32i.c` implements the RISC-V CPU emulation (i.e., how it emulates many different RISC-V instructions without running on a real RISC-V CPU). Your answer should be less than 40 words.

6. Compare the instructions counts between `bubble_iter` and `bubble_recur`. What's the notable differences between these two?

7. Why do you think the differences in Q6 is observed? Relate your answer with the differences between iteration and recursion.