MIPS Coding

Review

- Everything is stored in the computer as sequences of 0s and 1s
- Each assembly instruction is uniquely mapped to a unique sequence of 0s and 1s
- There are three types of instruction types in MIPS:
  - R-Types: opcode, rs, rt, rd, shamt, funt
  - I-Types: opcode, rs, rt, immediate
  - J-Types: opcode, immediate

Review

- opcode (6 bits): defines the operation
- rs/rt/rd (5 bits): register names / address
- shamt (5 bits): amount to shift in sll/srl
- funct (6 bits): further defines R-Types
- immediate (16 bits for I-Type / 26 for J-Type): addresses and constants

Exercise – the bubble sort

```
for (int i = 0; i < N-1; i++)
{
        for (int j = 0; j < N-i-1; j++)
        {
                if (A[j] < A[j+1])
                        swap(A[j], A[j+1]);
        }
}
```

Exercise – the bubble sort

- Need two loops – just encapsulate one in the other
- Need to read the elements – done before.
- Need to compare two numbers – done before
- Need to swap – not that hard

```
          .data
A:        .word 12, 34, 67, 1, 45, 90, 11, 33, 67, 19

          .text
          .globl main
main:
          la $s7, A                     # Address of A
          li $s6, 9                     # N-1

done:     li $v0,10
          syscall
```

Setup the program

```
          .data
A:        .word 12, 34, 67, 1, 45, 90, 11, 33, 67, 19

          .text
          .globl main
main:
          la $s7, A                     # getting the address
          li $s6, 9                     # N-1

          li $s0, 0                     # i = 0
LOOP1:

          addi $s0, $s0, 1              # i = i + 1
          bne $s0, $s6, LOOP1           # if i != N-1, outer loop again

done:     li $v0,10
          syscall
```

Getting the first loop done

```
          .data
A:        .word 12, 34, 67, 1, 45, 90, 11, 33, 67, 19

          .text
          .globl main
main:
          la $s7, A                     # getting the address
          li $s6, 9                     # N-1

          li $s0, 0                     # i = 0
LOOP1:    li $s1, 0                     # j = 0
LOOP2:

          addi $s1, $s1, 1              # j = j + 1
          sub $t7, $s6, $s0             # $t7 will get N-1-i
          bne $s1, $t7, LOOP2           # if j != N-1-i, inner loop again
          addi $s0, $s0, 1              # i = i + 1
          bne $s0, $s6, LOOP1           # if i != N-1, outer loop again

done:     li $v0,10
          syscall
```

Getting both loop done

```
          .data
A:        .word 12, 34, 67, 1, 45, 90, 11, 33, 67, 19

          .text
          .globl main
main:
          la $s7, A                     # getting the address
          li $s6, 9                     # N-1

          li $s0, 0                     # i = 0
LOOP1:    li $s1, 0                     # j = 0
LOOP2:    sll $t0, $s1, 2               # $t0 = j * 4
          add $t0, $t0, $s7             # $t0 is the address of  A[j]
          lw $t1, 0($t0)                # $t1 = A[j]
          lw $t2, 4($t0)                # $t2 = A[j+1]

          addi $s1, $s1, 1              # j = j + 1
          sub $t7, $s6, $s0             # $t7 will get N-1-i
          bne $s1, $t7, LOOP2           # if j != N-1-i, inner loop again
          addi $s0, $s0, 1              # i = i + 1
          bne $s0, $s6, LOOP1           # if i != N-1, outer loop again

done:     li $v0,10
          syscall
```

Adding the code to read the elements A[j] and A[j+1]

```
          .data
A:        .word 12, 34, 67, 1, 45, 90, 11, 33, 67, 19

          .text
          .globl main
main:
          la $s7, A                       # getting the address
          li $s6, 9                       # N-1

          li $s0, 0                       # i = 0
LOOP1:    li $s1, 0                       # j = 0
LOOP2:    sll $t0, $s1, 2                 # $t0 = j * 4
          add $t0, $t0, $s7               # $t0 is the address of  A[j]
          lw $t1, 0($t0)                  # $t1 = A[j]
          lw $t2, 4($t0)                  # $t2 = A[j+1]
          bgt $t1, $t2, L1                # if A[j] > A[j+1] goto L1, bypass the swapping
          sw $t1,   4($t0)                # do the swap
          sw $t2,   0($t0)                # do the swap
L1:       addi $s1, $s1, 1                # j = j + 1
          sub $t7, $s6, $s0               # $t7 will get N-1-i
          bne $s1, $t7, LOOP2             # if j != N-1-i, inner loop again
          addi $s0, $s0, 1                # i = i + 1
          bne $s0, $s6, LOOP1             # if i != N-1, outer loop again

done:     li $v0,10
          syscall
```

Adding the comparison and swapping

Pseudo instruction

- A pseudo instruction is not a real instruction supported by the hardware. It is created to make the coding easier. It is mapped to a **unique** sequence of real instructions by the assembler.

- blt $t0, $t1, L1
  - slt $at, $t0, $t1
  - bne $at, $0, L1
- bgt $t0, $t1, L1
  - slt $at, $t1, $t0
  - bne $at, $0, L1
- ble $t0, $t1, L1
  - slt $at, $t1, $t0
  - beq $at, $0, L1
- bge $t0, $t1, L1
  - slt $at, $t0, $t1
  - beq $at, $0, L1

- li/la $t0, 0x3BF20
  - lui $t0, 0x0003
  - ori $t0, $0, 0xBF20
- not $t0, $s0
  - nor $t0, $s0, $0
- move $t0, $t1
  - ori $t0, $t1, $0

- http://www.utdallas.edu/~cantrell/ee2310/spim.inst.txt

In-class exercise -- Loop