Exercise #1

- Represent $+7.25_{10}$ in binary single precision

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
Exercise #2

• What decimal number is represented by this single precision float?

0100 0001 0110 1000 0000 0000 0000 0000

Exercise #3

• Represent -17.5 in binary single precision and double precision
MIPS Floating Point Basics

- Floating point registers
  $f0, f1, f2, ..., f31
  Used in pairs for double precision (f0, f1) (f2, f3), ...
  $f0 not always zero

- Register conventions:
  - Function arguments passed in
  - Function return value stored in
  - Where are addresses (e.g. for arrays) passed?

- Load and store:
  lwc1 $f2, 0($sp)
  swc1 $f4, 4($t2)

MIPS FP Arithmetic

- Addition, subtraction: add.s, add.d, sub.s, sub.d
  add.s $f1, $f2, $f3
  add.d $f2, $f4, $f6

- Multiplication, division: mul.s, mul.d, div.s, div.d
  mul.s $f2, $f3, $f4
  div.s $f2, $f4, $f6

MIPS FP Control Flow

- Pattern of a comparison: c.___.s (or c.___.d)
  c.lt.s $f2, $f3
  c.ge.d $f4, $f6

- Where does the result go?

- Branching:
  bclt label10
  bclf label20

Example #1

- Convert the following C code to MIPS:
  float max (float A, float B) {
    if (A <= B) return A;
    else return B;
  }
Example #2

- Convert the following C code to MIPS:

```c
void setArray (float F[], int index,
               float val) {
    F[index] = val;
}
```

Exercise #1

- Convert the following C code to MIPS:

```c
float pick (float G[], int index) {
    return G[index];
}
```

Exercise #2

- Convert the following C code to MIPS:

```c
float max (float A, float B) {
    if (A > B) return A / B;
    else return B / A;
}
```
Exercise #3

- Convert the following C code to MIPS:
  ```c
  float sum (float A[], int N) {
    int j;
    float sum = 0.0;
    for (j=0; j<N; j++)
      sum = sum + A[j]
    return sum;
  }
  ```

Exercise #4 – Stretch

- Convert the following C code to MIPS:
  ```c
  float average (float A[], int N) {
    int j;
    float sum = 0.0;
    for (j=0; j<N; j++)
      sum = sum + A[j]
    return sum / N;
  }
  ```
Chapter Four Summary

- Computer arithmetic is constrained by limited precision
- Bit patterns have no inherent meaning but standards do exist
  - two’s complement
  - IEEE 754 floating point
- Computer instructions determine “meaning” of the bit patterns
- Performance and accuracy are important so there are many complexities in real machines (i.e., algorithms and implementation).

- We are ready to move on (and implement the processor)

Chapter Goals

- Introduce 2’s complement numbers
  - Addition and subtraction
  - Sketch multiplication, division
- Overview of ALU (arithmetic logic unit)
- Floating point numbers
  - Representation
  - Arithmetic operations
  - MIPS instructions