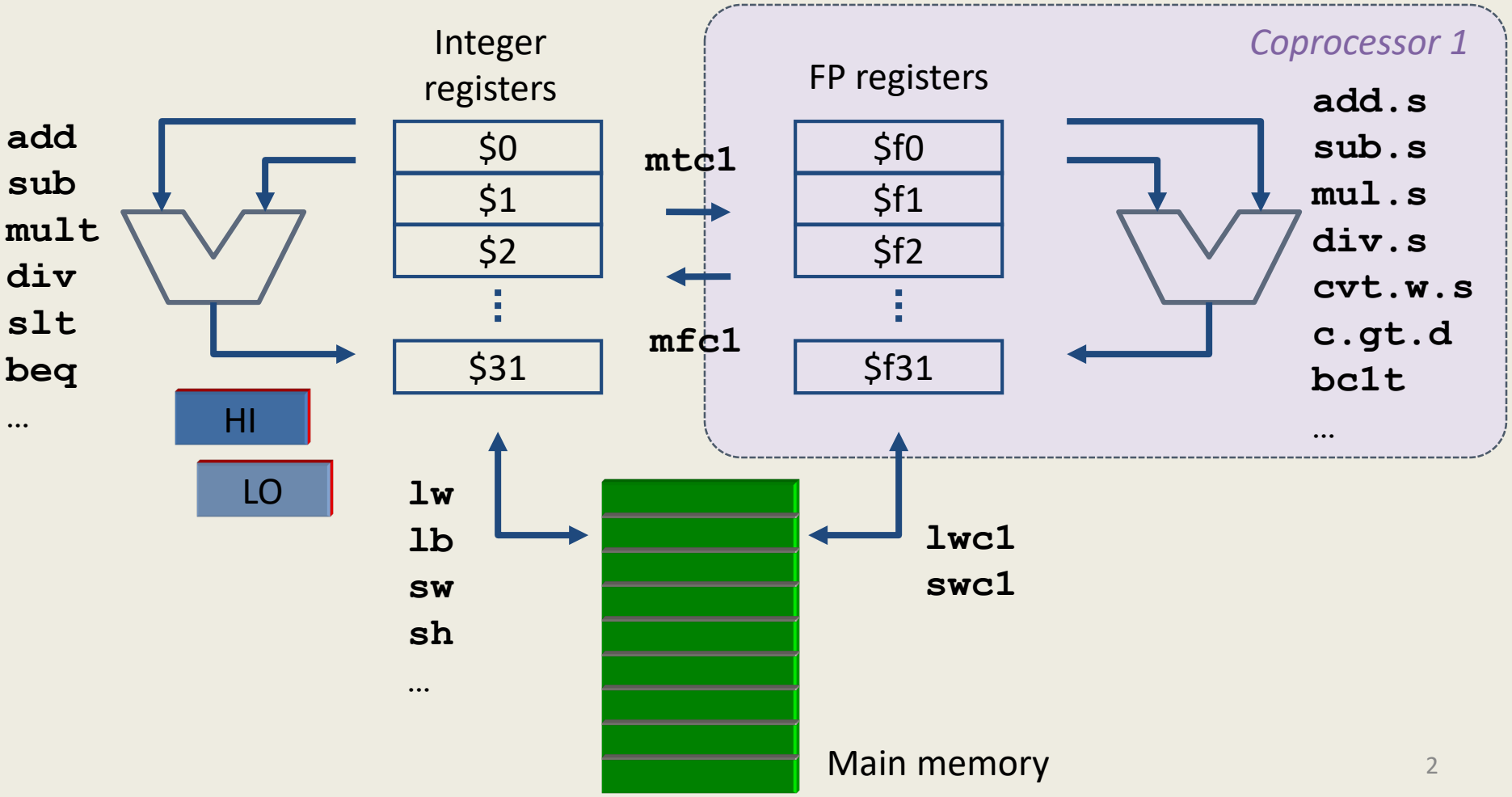
Lab Session 7

Floating Point Arithmetic

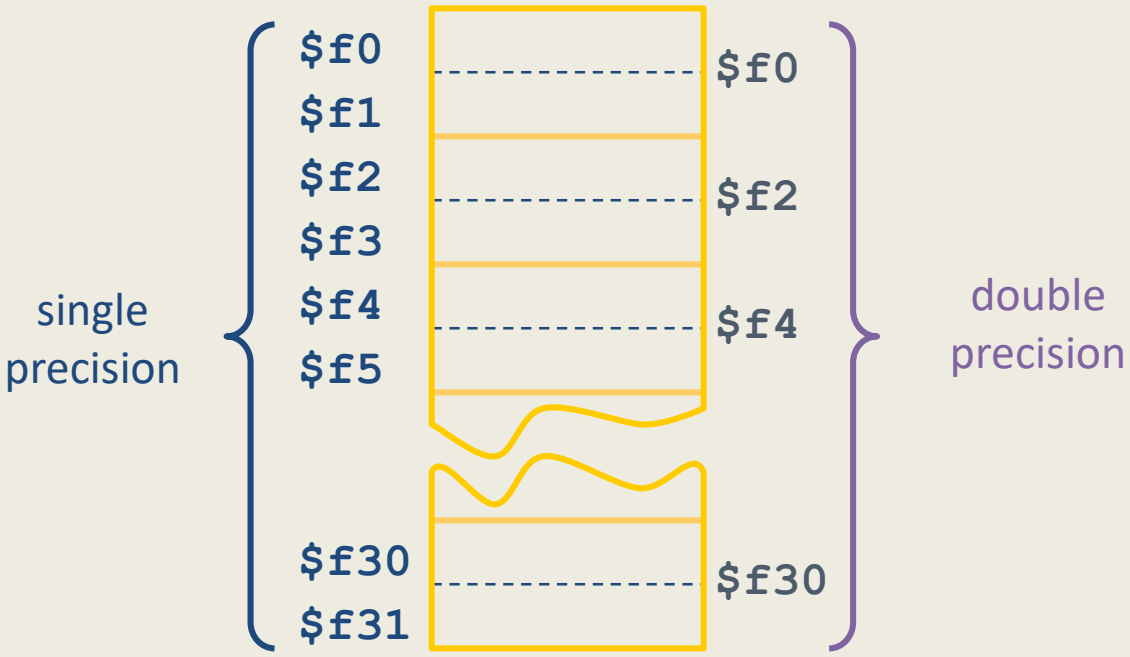
FP in MIPS

Programmer's view



FP register file

- 32 registers named $f0, $f1, … $f31 for 32-bit float values
- They can be paired to hold a 64-bit value (double)
  - If $f0 contains a double, then $f0 holds the least significant half and $f1 the most significant part



Register use convention

Name of register	Conventional use
$f0	Function return (real part)
$f2	Function return (imaginary part)
$f4,$f6,$f8,$f10	Temporary registers
$f12,$f14	Function parameters
$f16,$f18	Temporary registers
$f20,$f22,$f24,$f26,$f28,$f30	Registers to preserve among function calls

Data transfer instructions

- Data interchange with memory and integer registers

operation	instruction
read: *$ft ← Mem[X+$rs]*	`lwc1 $ft,X($rs)`
write: *Mem[X+$rs] ← $ft*	`swc1 $ft,X($rs)`
transfer: *$ft ← $rs*	`mtc1 $rs,$ft`
transfer: *$rt ← $fs*	`mfc1 $rt,$fs`

fs, ft: FP registers

rs, rt: Int registers

```
        .data
x:      .float 3.1416
y:      .double 0.1
        .text
        la $t0,x
        lwc1 $f0,0($t0) # f0 <- x
        la $t0,y
        lwc1 $f2,0($t0)
        lwc1 $f3,4($t0) # f2 <- y
        mtc1 $0,$f4     # f4 <- 0.0
```

FP instructions do not handle immediate operands. Constants must be allocated in memory or built into integer registers and then moved

Type conversion

- FP registers may contain
  - **`s`**: Single-precision FP values
  - **`d`**: Double-precision FP values
  - **`w`**: 32-bit integer values
- Type conversion is possible via `cvt._._ fd,fs`
  - Eg., `cvt.d.w $f4,$f7` converts the integer in f7 into a double in f4
- In combination with transfers to-from integer registers, values of different types can be used in arithmetic expressions

Basic arithmetic operations

- Each operation has S and D versions (single and double)
  - Eg., `add.s $f0,$f1,$f2` vs. `add.d $f0,$f2,$f4`

operation	instruction
addition	**`add._ fd,fs,ft`**
subtraction	**`sub._ fd,fs,ft`**
multiplication	**`mul._ fd,fs,ft`**
division	**`div._ fd,fs,ft`**
comparison	**`c.`*`cond`*`._ fs,ft`**
copy	**`mov._ fd,fs`**
sign change	**`neg._ fd,fs`**
absolute value	**`abs._ fd,fs`**

Immediate load pseudoinstructions

```
li.s $f0, 5.678
li.d $f4, 9.012
```

Comparison instructions

`c.cond._ fs,ft`

- Comparison instructions store their result in bit **FPc**
  - TRUE = 1; FALSE = 0
- FPc is kept in a control register of coprocessor 1 and is used by conditional branch instructions
- There is a set of comparison instructions for each data type
- Eg., `c._.s fd, fs` or `c._.d fd, fs`

fd>fs	fd=fs	fd<fs
`gt`	`eq`	`lt`
`le`	`neq`	`ge`
fd≤fs	fd≠fs	fd≥fs

Flow control

- Two conditional branch instructions:
  - **`bc1t`** *`label`* – if FPc = 1 then branch to *label*
  - **`bc1f`** *`label`* – if FPc = 0 then branch to *label*
- Combined with comparison instructions, they enable complex conditional branches
- Each condition accepts two implementations
  - SP example: ***if*** *($f0 > $f2)* ***then*** *branch to label*

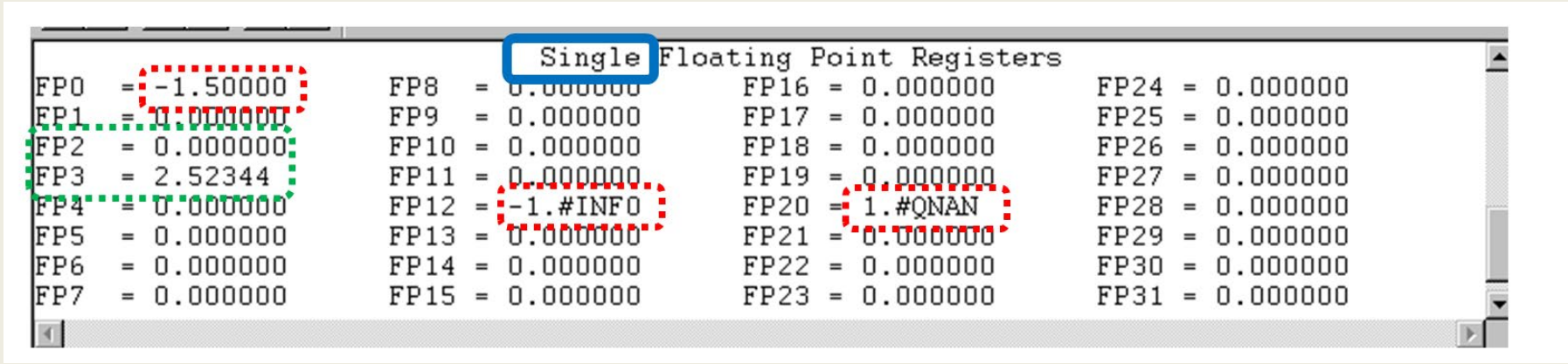
```
; check wether $f0>$f2
      c.gt.s $f0,$f2
; branch if true
      bc1t label
```

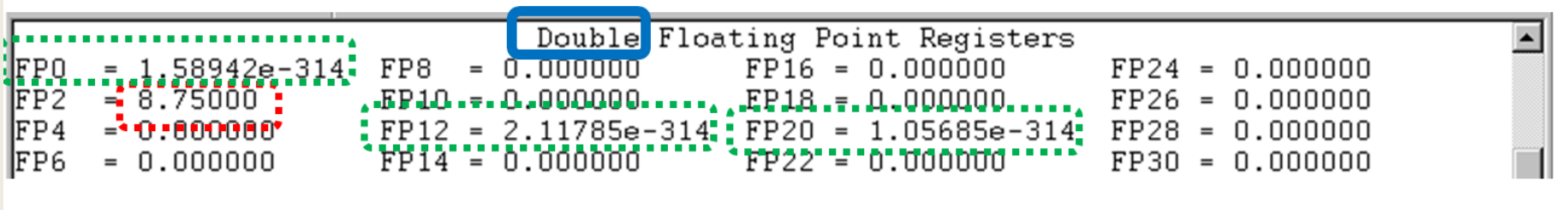
```
; check wether $f0<=$f2
      c.le.s $f0,$f2
; branch if false
      bc1f label
```

Lab Exercise 1

- Check registers

Use the program: *formatos.s*





Lab Exercise 2

- **Arithmetic mean calculation**

Use the program: *promedio.s*

Analyze the program and check results in single and double precision

Lab Exercise 3

- **Number π calculation**

Use the program: *pi-leibniz.s*

1. Analyze the program and obtain results for different number of iterations
2. Adapt the program to double precision numbers. *pi-leibniz-d.s.*