Function Calls

- **Caller:** calling function (in this case, `main`)
- **Callee:** called function (in this case, `sum`)

C Code

```
void main()
{
  int y;
  y = sum(42, 7);
  ...
}

int sum(int a, int b)
{
  return (a + b);
}
```

Function Conventions

- **Caller:**
  - passes **arguments** to callee
  - jumps to callee
- **Callee:**
  - **performs** the function
  - **returns** result to caller
  - **returns** to point of call
  - **must not overwrite** registers or memory needed by caller

MIPS Function Conventions

- **Call Function:** jump and link (`jal`)
- **Return** from function: jump register (`jr`)
- **Arguments**: `$a0 – $a3`
- **Return value**: `$v0`

Function Calls

C Code

```
int main() {
  simple();
  a = b + c;
}

void simple() {
  return;
}
```

MIPS assembly code

```
0x00400200 main: jal  simple
0x00400204       add  $s0, $s1, $s2
...

0x00401020 simple: jr $ra
```

`void` means that `simple` doesn't return a value

Function Calls

C Code

```
int main() {
  simple();
  a = b + c;
}

void simple() {
  return;
}
```

MIPS assembly code

```
0x00400200 main: jal  simple
0x00400204       add  $s0, $s1, $s2
...

0x00401020 simple: jr $ra
```

jal: jumps to `simple`
`$ra` = PC + 4 = 0x00400204

jr $ra: jumps to address in `$ra` (0x00400204)

Input Arguments & Return Value

MIPS conventions:

- Argument values: `$a0` - `$a3`
- Return value: `$v0`

Input Arguments & Return Value

C Code

```
int main()
{
  int y;
  ...
  y = diffofsums(2, 3, 4, 5);  // 4 arguments
  ...
}

int diffofsums(int f, int g, int h, int i)
{
  int result;
  result = (f + g) - (h + i);
  return result;                // return value
}
```

Input Arguments & Return Value

MIPS assembly code

```
# $s0 = y

main:
  ...
  addi $a0, $0, 2     # argument 0 = 2
  addi $a1, $0, 3     # argument 1 = 3
  addi $a2, $0, 4     # argument 2 = 4
  addi $a3, $0, 5     # argument 3 = 5
  jal  diffofsums     # call Function
  add  $s0, $v0, $0   # y = returned value
  ...

# $s0 = result
diffofsums:
  add $t0, $a0, $a1   # $t0 = f + g
  add $t1, $a2, $a3   # $t1 = h + i
  sub $s0, $t0, $t1   # result = (f + g) - (h + i)
  add $v0, $s0, $0    # put return value in $v0
  jr  $ra             # return to caller
```

Input Arguments & Return Value

MIPS assembly code

```
# $s0 = result
diffofsums:
  add $t0, $a0, $a1   # $t0 = f + g
  add $t1, $a2, $a3   # $t1 = h + i
  sub $s0, $t0, $t1   # result = (f + g) - (h + i)
  add $v0, $s0, $0    # put return value in $v0
  jr  $ra             # return to caller
```

- `diffofsums` overwrote 3 registers: `$t0`, `$t1`, `$s0`
- `diffofsums` can use *stack* to temporarily store registers