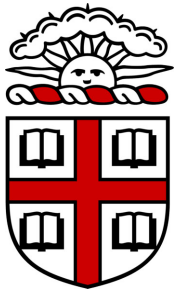
EN164: Design of Computing Systems
Lecture 11: Processor / ISA 4

Professor Sherief Reda
http://scale.engin.brown.edu
Electrical Sciences and Computer Engineering
School of Engineering
Brown University
Spring 2011



[material from Patterson & Hennessy, 4th ed and Harris 1st ed]

Procedure calls

Procedure calling conventions:

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

MIPS conventions:

- Call procedure: jump and link (`jal`)
- Return from procedure: jump register (`jr`)
- Argument values: $a0 - $a3
- Return value: $v0 -$v1

Procedure calls

High-level 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.

`jal:` jumps to `simple` and saves `PC`+4 in the return address register (`$ra`). In this case, `$ra` = 0x00400204 after `jal` executes.

`jr $ra:` jumps to address in `$ra`, in this case 0x00400204.

Input arguments and return values

High-level 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 and return values

MIPS assembly code

```
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 procedure
   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
```

MIPS conventions:

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

Potential problems in procedural calling

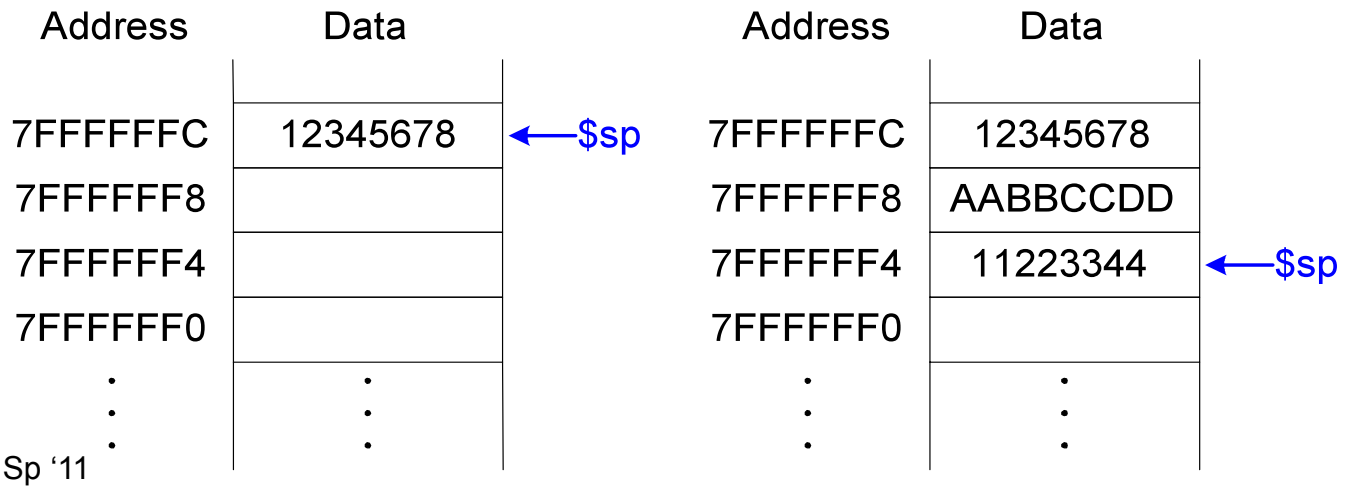
MIPS assembly code

```
# $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`, and `$s0`
- `diffofsums` can use the *stack* to temporarily store registers

The stack

- Memory used to temporarily save variables
- Like a stack of dishes, last-in-first-out (LIFO) queue
- *Expands*: uses more memory by growing down (from higher to lower memory addresses) when more space is needed
- *Contracts*: uses less memory when the space is no longer needed
- Stack pointer: `$sp`, points to top of the stack



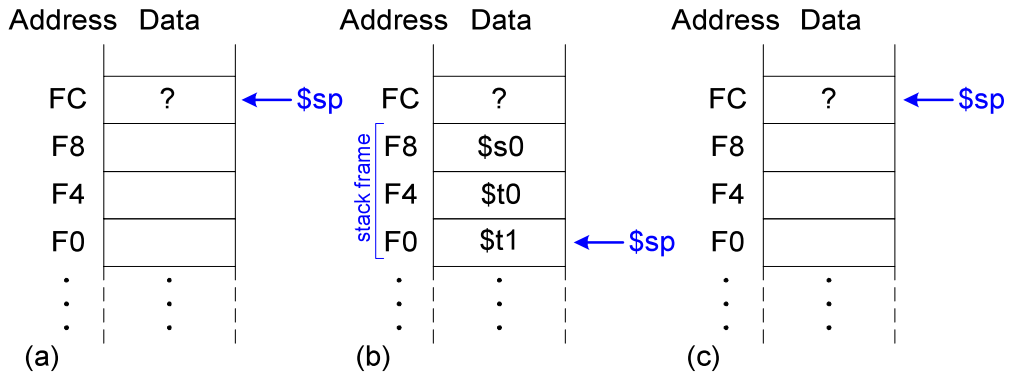
How procedures use the stack

- Called procedures must have no other unintended side effects.
- But `diffofsums` overwrites 3 registers: `$t0`, `$t1`, `$s0`

```
# MIPS assembly
# $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
```

Storing register values on the stack

```
# $s0 = result
diffofsums:
  addi $sp, $sp, -12   # make space on stack
                       # to store 3 registers
  sw   $s0, 8($sp)     # save $s0 on stack
  sw   $t0, 4($sp)     # save $t0 on stack
  sw   $t1, 0($sp)     # save $t1 on stack
  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
  lw   $t1, 0($sp)     # restore $t1 from stack
  lw   $t0, 4($sp)     # restore $t0 from stack
  lw   $s0, 8($sp)     # restore $s0 from stack
  addi $sp, $sp, 12    # deallocate stack space
  jr   $ra             # return to caller
```



Protocol for preserving registers

Preserved *Callee-Saved*	Nonpreserved *Caller-Saved*
`$s0 - $s7`	`$t0 - $t9`
`$ra`	`$a0 - $a3`
`$sp`	`$v0 - $v1`
stack above `$sp`	stack below `$sp`

Multiple procedure calls

```
proc1:
  addi $sp, $sp, -4    # make space on stack
  sw   $ra, 0($sp)     # save $ra on stack
  jal  proc2
  ...
  lw   $ra, 0($sp)     # restore $s0 from stack
  addi $sp, $sp, 4     # deallocate stack space
  jr  $ra              # return to caller
```

Storing saved registers on the stack

```
# $s0 = result
diffofsums:
  addi $sp, $sp, -4   # make space on stack to
                      # store one register
  sw  $s0, 0($sp)     # save $s0 on stack
                      # no need to save $t0 or $t1
  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
  lw  $s0, 0($sp)     # restore $s0 from stack
  addi $sp, $sp, 4    # deallocate stack space
  jr  $ra             # return to caller
```

Recursive procedure calls

High-level code

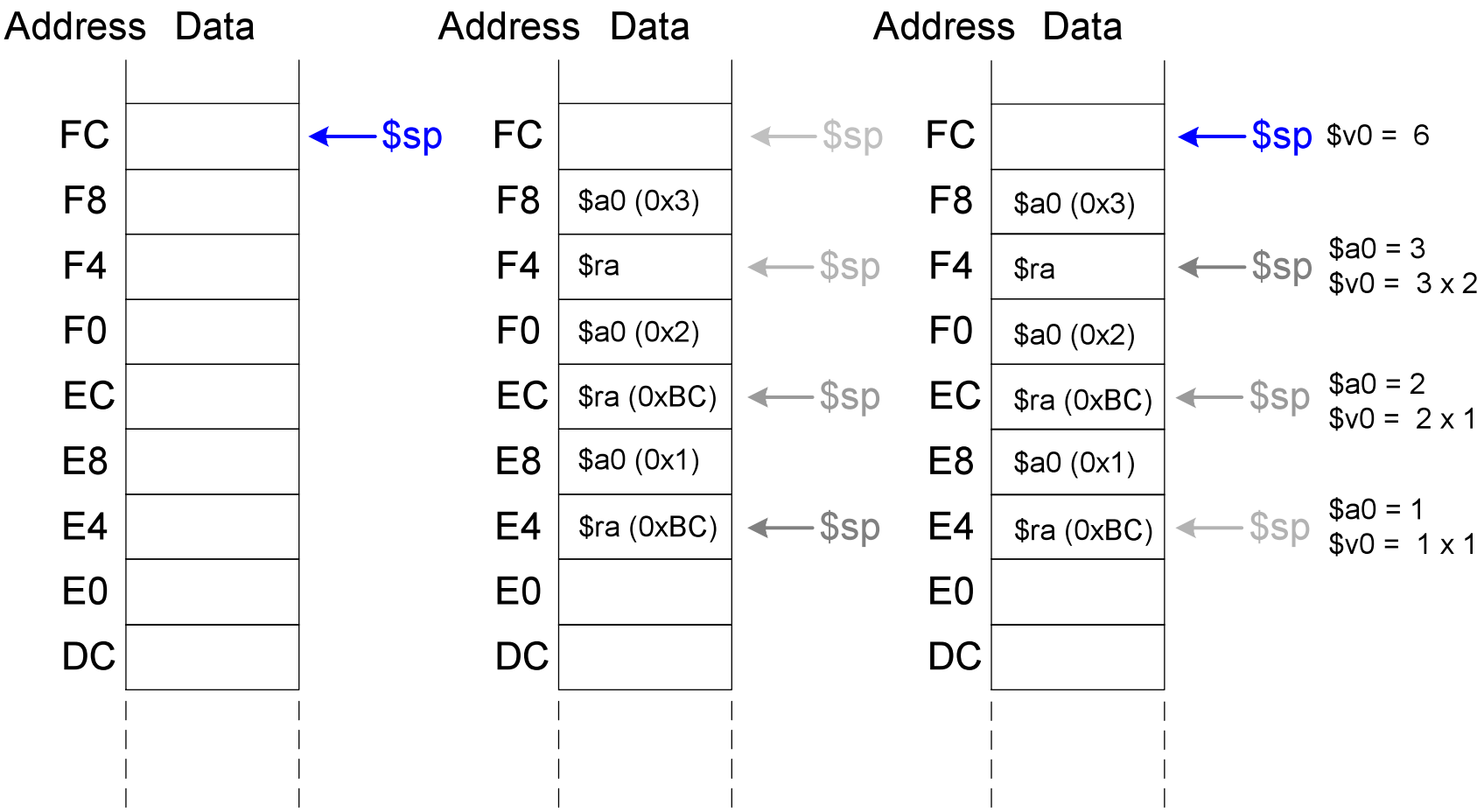
```
int factorial(int n) {
  if (n <= 1)
    return 1;
  else
    return (n * factorial(n-1));
}
```

Recursive procedure calls

MIPS assembly code

```
0x90 factorial: addi $sp, $sp, -8   # make room
0x94            sw   $a0, 4($sp)    # store $a0
0x98            sw   $ra, 0($sp)    # store $ra
0x9C            addi $t0, $0, 2
0xA0            slt  $t0, $a0, $t0  # a <= 1 ?
0xA4            beq  $t0, $0, else  # no: go to else
0xA8            addi $v0, $0, 1     # yes: return 1
0xAC            addi $sp, $sp, 8    # restore $sp
0xB0            jr   $ra            # return
0xB4       else: addi $a0, $a0, -1  # n = n - 1
0xB8            jal  factorial      # recursive call
0xBC            lw   $ra, 0($sp)    # restore $ra
0xC0            lw   $a0, 4($sp)    # restore $a0
0xC4            addi $sp, $sp, 8    # restore $sp
0xC8            mul  $v0, $a0, $v0  # n * factorial(n-1)
0xCC            jr   $ra            # return
```

Stack during recursive calls



Procedural call summary

- Caller
  - Put arguments in $a0-$a3
  - Save any registers that are needed ($ra, maybe $t0-t9)
  - jal callee
  - Restore registers
  - Look for result in $v0
- Callee
  - Save registers that might be disturbed ($s0-$s7)
  - Perform procedure
  - Put result in $v0
  - Restore registers
  - jr $ra