立刻移除水印

Procedures

Assignment Project Exam Help

https://powcoder.com

pdfelement
万兴PDF专家

Add WeChat powcoder

Overview

- C Functions
- MIPS Instructions for Procedures
- The Stack
- Register Conventions
- Another Example

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder
pdfelement
万兴PDF专家
立刻移除水印

C functions

```
main() {
 int i,j,k,m;

 i = mult(j,k); ... ;
 m = mult(i,i); ...

}

/* really dumb mult function */

int mult (int mcand, int mlier){
 int product;

 product = 0;
 while (mlier > 0)  {
  product = product + mcand;
  mlier = mlier -1; }
 return product;

}
```

What information must compiler/programmer keep track of?

What instructions can accomplish this?

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder
pdfelement
万兴PDF专家
立刻移除水印

Function Call Bookkeeping

立刻移除水印

- Registers play a major role in keeping track of information for function calls.
- Register conventions:
  - Return address `$ra`
  - Arguments `$a0, $a1, $a2, $a3`
  - Return value `$v0, $v1`
  - Local variables `$s0, $s1, … , $s7`
- The stack is also used.
- More on this later.

Assignment Project Exam Help
pdfelement
https://powcoder.com
万兴PDF专家
Add WeChat powcoder

Instruction Support for Functions (1/4)

立刻移除水印

C

```
... sum(a,b);... /* a,b:$s0,$s1 */
}
int sum(int x, int y) {
    return x+y;
}
```

Assignment Project Exam Help

MIPS

https://powcoder.com

Add WeChat powcoder

```
address
1000 add   $a0,$s0,$zero   # x = a
1004 add   $a1,$s1,$zero   # y = b
1008 addi  $ra,$zero,1016 #$ra=1016
1012 j     sum            #jump to sum
1016 ...

2000 sum:  add $v0,$a0,$a1
2004 jr    $ra  # new instruction
```

Instruction Support for Functions (2/4)

- Single instruction to jump and save return address: jump and link (`jal`)
- Before:

```
1008 addi $ra,$zero,1016 #$ra=1016
1012 j sum               #go to sum
```

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder
pdfelement 万兴PDF专家
立刻移除水印

- After:

```
1012 jal sum  # $ra=1016,go to sum
```

- Why have a `jal`? Make the common case fast: functions are very common.

Instruction Support for Functions (3/4)

- Syntax for `jal` (jump and link) is same as for `j` (jump):

```
jal label
```

Assignment Project Exam Help

- `jal` should really be called `laj` for "link and jump":

pdfelement 万兴PDF专家
https://powcoder.com
Add WeChat powcoder

  - Step 1 (link): Save address of *next* instruction into $ra (Why next instruction? Why not current one?)
  - Step 2 (jump): Jump to the given label

立刻移除水印

立刻移除水印

Instruction Support for Functions (4/4)

- Syntax for `jr` (jump register):

```
jr   register
```

- Instead of providing a label to jump to, the `jr` instruction provides a register which contains an address to jump to.

Assignment Project Exam Help
https://powcoder.com
pdfelement
万兴PDF专家
Add WeChat powcoder

- Only useful if we know exact address to jump to: rarely applicable.
- Very useful for function calls:
  - `jal` stores return address in register ($ra)
  - `jr` jumps back to that address

Nested Procedures (1/2)

```
int sumSquare(int x, int y) {
    return mult(x,x)+ y;
 }
```

- Something called `sumSquare`, now `sumSquare` is calling `mult`.
- So there's a value in $ra that `sumSquare` wants to jump back to, but this will be overwritten by the call to `mult`.
- Need to save `sumSquare` return address before call to `mult`.

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder
立刻移除水印

Nested Procedures (2/2)

立刻移除水印

- In general, may need to save some other info in addition to $ra.

- When a C program is run, there are 3 important memory areas allocated:

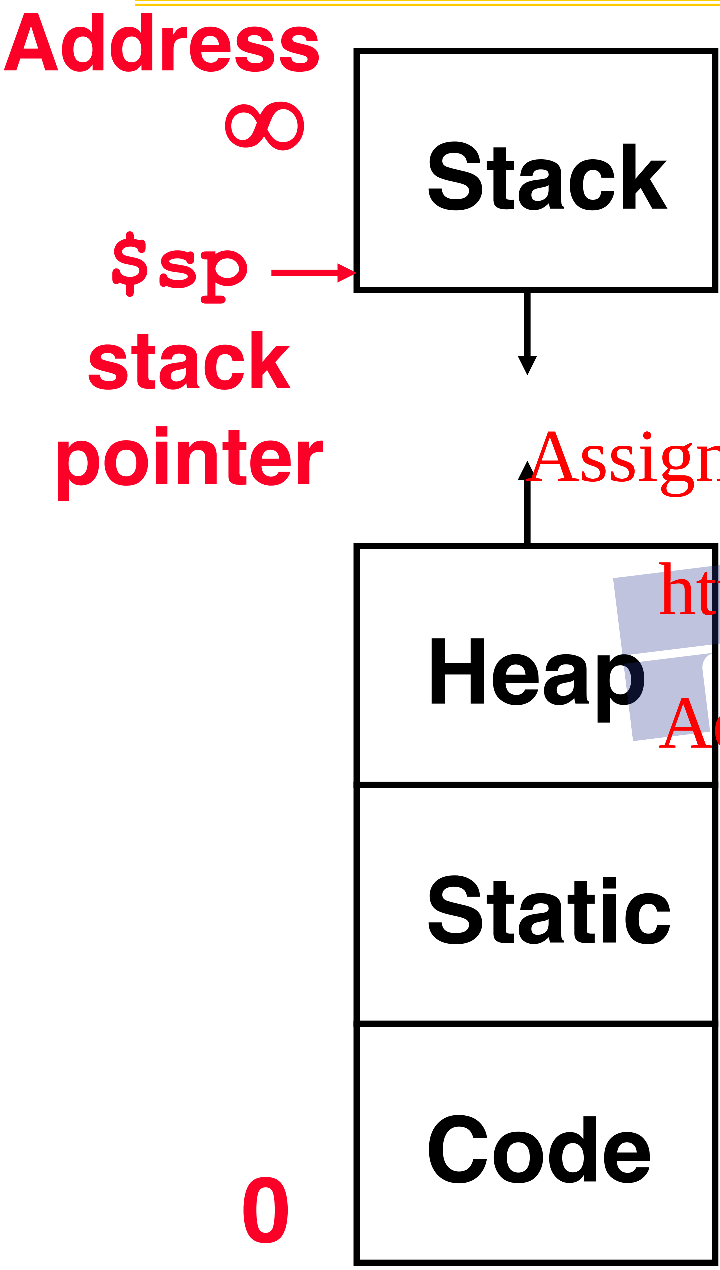
Assignment Project Exam Help

  - Static: Variables declared once per program, cease to exist only after execution completes

https://powcoder.com
pdfelement
万兴PDF专家
Add WeChat powcoder

  - Heap: Variables declared dynamically
  - Stack: Space to be used by procedure during execution; this is where we can save register values

C memory Allocation



Space for saved procedure information

Explicitly created space, e.g., malloc(); C pointers

Variables declared once per program

Program

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder
pdfelement
万兴PDF专家
立刻移除水印

立刻移除水印

Using the Stack (1/2)

- So we have a register $sp which always points to the last used space in the stack.
- To use stack, we decrement this pointer by the amount of space we need and then fill it with info.
- So, how do we compile this?

```
int sumSquare(int x, int y) {
  return mult(x,x)+ y;
}
```

Assignment Project Exam Help
https://powcoder.com
pdfelement
万兴PDF专家
Add WeChat powcoder

Using the Stack (2/2)

立刻移除水印

° **Compile by hand**

```
sumSquare:
        addi $sp,$sp,-8 #space on stack
        sw $ra, 4($sp)   # save ret addr
        sw $a1, 0($sp)   # save y

        add $a1,$a0,$zero # mult(x,x)
        jal mult         # call mult

        lw $a1, 0($sp)   # restore y
        add $v0,$v0,$a1 # mult()+y
        lw $ra, 4($sp)   # get ret addr
        addi $sp,$sp,8 # restore stack
        jr $ra
```

Assignment Project Exam Help
pdfelement
https://powcoder.com
万兴PDF专家
Add WeChat powcoder

Steps for Making a Procedure Call

立刻移除水印

1) Save necessary values onto stack.

2) Assign argument(s), if any.

3) `jal` call

4) Restore values from stack.

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder
pdfelement
万兴PDF专家

Rules for Procedures

- Called with a `jal` instruction, returns with a `jr $ra`
- Accepts up to 4 arguments in \$a0, \$a1, \$a2 and \$a3
- Return value is always in \$v0 (and if necessary in \$v1)
- Must follow **register conventions** (even in functions that only you will call)! So what are they?

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder
pdfelement
万兴PDF专家
立刻移除水印

MIPS Registers (1/2)

The constant 0	$0	$zero
Reserved for Assembler	$1	$at
Return Values	$2-$3	$v0-$v1
Arguments	$4-$7	$a0-$a3
Temporary	$8-$15	$t0-$t7
Saved	$16-$23	$s0-$s7
More Temporary	$24-$25	$t8-$t9

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder
pdfelement
万兴PDF专家
立刻移除水印

MIPS Registers (2/2)

Used by Kernel	$26-27	$k0-$k1
Global Pointer	$28	$gp
Stack Pointer	$29	$sp
Frame Pointer	$30	$fp
Return Address	$31	$ra

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder
pdfelement 万兴PDF专家
立刻移除水印

- In general, feel free to use either the name or the number, but try not to use both within the same piece of code.
- We prefer names, they make code more readable.

Register Conventions (1/5)

- Caller: the calling function
- Callee: the function being called
- When callee returns from executing, the caller needs to know which registers may have changed and which are guaranteed to be unchanged.

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

- **Register Conventions**: A set of generally accepted rules as to which registers will be unchanged after a procedure call (`jal`) and which may be changed.

立刻移除水印

Register Conventions (2/5)

- $0: No Change. Always 0.
- $v0-$v1: Change. These are expected to contain new values.
- $a0-$a3: Change. These are volatile argument registers.
- $t0-$t9: Change. That's why they're called temporary: any procedure may change them at any time.

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder
pdfelement
万兴PDF专家
立刻移除水印

Register Conventions (3/5)

- $s0-$s7: **No Change**. Very important, that's why they're called saved registers. If the callee changes these in any way, it must restore the original values before returning.

Assignment Project Exam Help

- $sp: **No Change**. The stack pointer must point to the same place before and after the jal call, or else the caller won't be able to restore values from the stack.

https://powcoder.com

Add WeChat powcoder

- $ra: Change. The `jal` call itself will change this register.

Register Conventions (4/5)

- What do these conventions mean?
  - If function A calls function B, then function A must save any temporary registers that it may be using onto the stack before making a `jal` call.
  - Function B must save any S (saved) registers it intends to use before garbling up their values
  - Remember: Caller/callee need to save only temporary/saved registers they are using, not all registers.

立刻移除水印
Assignment Project Exam Help
pdfelement
https://powcoder.com
万兴PDF专家
Add WeChat powcoder

Register Conventions (5/5)

- Note that, if the *callee* is going to use some s registers, it must:
  - save those s registers on the stack
  - use the registers
  - restore s registers from the stack
  - `jr $ra`
- With the temp registers, the callee doesn't need to save onto the stack.
- Therefore the *caller* must save those temp registers that it would like to preserve though the call.

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder
pdfelement
万兴PDF专家
立刻移除水印

Other Registers

- $at: may be used by the assembler at any time; unsafe to use
- $k0-$k1: may be used by the kernel at any time; unsafe to use
- $gp: don't worry about it
- $fp: don't worry about it
- Note: Feel free to read up on $gp and $fp in Appendix A, but you can write perfectly good MIPS code without them.

立刻移除水印
Assignment Project Exam Help
pdfelement
https://powcoder.com
万兴PDF专家
Add WeChat powcoder

立刻移除水印

Example: Compile This (1/5)

```
main() {
 int i,j,k,m; /* i-m:$s0-$s3 */

 i = mult(j,k); ... ;
 m = mult(i,i); ...

}

int mult (int mcand, int mlier){
 int product;

 product = 0;
 while (mlier > 0)  {
  product += mcand;
  mlier -= 1; }
 return product;

}
```

Assignment Project Exam Help

pdfelement 万兴PDF专家

https://powcoder.com

Add WeChat powcoder

Example: Compile This (2/5)

```
__start:

  add $a0,$s1,$0      # arg0 = j
  add $a1,$s2,$0      # arg1 = k
  jal mult            # call mult
  add $s0,$v0,$0      # i = mult()
  ...
  add $a0,$s0,$0      # arg0 = i
  add $a1,$s0,$0      # arg1 = i
  jal mult            # call mult
  add $s3,$v0,$0      # m = mult()
  ...
done
```

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder
pdfelement
万兴PDF专家
立刻移除水印

Example: Compile This (3/5)

- Notes:
  - `main` function ends with `done`, not `jr $ra`, so there's no need to save $ra onto stack
  - all variables used in `main` function are saved registers, so there's no need to save these onto stack

立刻移除水印

Assignment Project Exam Help

https://powcoder.com

万兴PDF专家

Add WeChat powcoder

Example: Compile This (4/5)

立刻移除水印

```
mult:
    add   $t0,$0,$0      # prod=0
Loop:
    slt   $t1,$0,$a1     # mlr > 0?
    beq   $t1,$0,Fin     # no=>Fin
    add   $t0,$t0,$a0    # prod+=mc
    addi  $a1,$a1,-1     # mlr-=1
    j     Loop           # goto Loop
Fin:
    add   $v0,$t0,$0     # $v0=prod
    jr    $ra            # return
```

Assignment Project Exam Help
pdfelement
https://powcoder.com
万兴PDF专家
Add WeChat powcoder

立刻移除水印

Example: Compile This (5/5)

- **Notes:**
  - **no `jal` calls are made from `mult` and we don't use any saved registers, so we don't need to save anything onto stack**
  - **temp registers are used for intermediate calculations (could have used s registers, but would have to save the caller's on the stack.)**
  - **$a1 is modified directly (instead of copying into a temp register) since we are free to change it**
  - **result is put into $v0 before returning**

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder
PDFelement
万兴PDF专家

Things to Remember (1/2)

立刻移除水印

- Functions are called with `jal`, and return with `jr $ra`.

- The stack is your friend: Use it to save anything you need. Just be sure to leave it the way you found it.

Assignment Project Exam Help
pdfelement
https://powcoder.com
万兴PDF专家
Add WeChat powcoder

- Register Conventions: Each register has a purpose and limits to its usage. Learn these and follow them, even if you're writing all the code yourself.

Things to Remember (2/2)

- **Instructions we know so far**

  Arithmetic: `add, addi, sub, addu, addiu, subu, sll`

  Memory: `lw, sw`

  Decision: `beq, bne, slt, slti, sltu, sltiu`

  Unconditional Branches (Jumps): `j, jal, jr`

- **Registers we know so far**
  - All of them!

立刻移除水印
Assignment Project Exam Help
pdfelement
https://powcoder.com
万兴PDF专家
Add WeChat powcoder