CSEE W3827

Fundamentals of Computer Systems

Homework Assignment 3

Solutions

Prof. Stephen A. Edwards

Columbia University

Due June 22, 2016 at 5:30 PM

Name: Solutions

Uni:

Show your work for each problem; we are more interested in how you get the answer than whether you get the right answer.

1. (20 pts.) In MIPS assembly, implement the standard C function `rindex`:

   `char *rindex(const char *s, int c)`

   This returns a pointer to the *rightmost* occurrence of the character *c* in the string *s* or NULL if the character is not found. The terminating null byte is considered to be part of the string.

   Start from the `rindex.s` template on the class website; use the SPIM simulator.

   Your function must obey MIPS calling conventions.

   Turn in your solution on paper with evidence that it works. Add some test cases. Also, upload your solution as a single .s file to Courseworks.

   On the supplied test harness, your code should print

```
Looking for ’e’ in "Hello World!"
Found at position 1
Looking for ’l’ in "Hello World!"
Found at positio<n 9
Looking for ’z’ in "Hello World!"
Not found
Looking for ’Hello World!"
Found at position 12
Looking for ’z’ in "The quick brown fox jumps over the lazy dog"
Found at position 37
```

```
    # $a0 : s
    # $a1 : c
rindex:
    move $v0, $0      # Default: did not find
loop:
    lb $t0, 0($a0)    # Get the character
    bne $t0, $a1, rindex_not
    move $v0, $a0     # It matched: remember its address
rindex_not:
    addiu $a0, $a0, 1 # Go to the next character
    bne $t0, $0, loop # Not at the terminating 0?  Go again
    jr $ra
```

2. (30 pts.) In MIPS assembly, implement an "eval" function that walks a binary tree that represents an arithmetic expression and computes its meaning. Each tree node begins with a byte that indicates the the node is an integer (leaf) or operator plus two pointers to their arguments. In C, this would be

```c
struct expr {
  char op; /* 0 for leaf */
  union {
    int leaf;
    struct {
      struct expr *left, *right;
    } branch;
  } pl;
};
```

```c
int eval(struct expr *e)
{
  int left, right;
  if (e->op == 0) return e->pl.leaf;
  left = eval(e->pl.branch.left);
  right = eval(e->pl.branch.right);
  switch (e->op) {
  case '+': return left + right;
  case '-': return left - right;
  case '*': return left * right;
  }
  return 0;
}
```

Start from the `eval.s` template on the class website.

Your function must obey MIPS calling conventions. Use the stack to implement the recursion.

Implement your function in the SPIM simulator.

Turn in your solution on paper with evidence that it works. Add some test cases. Also, upload your solution as a single .s file to Courseworks.

On the supplied test harness, your code should print

```
42 = 42
17 = 17
25 = 25
(17+25) = 42
(5*(2+3)) = 25
((5*(2+3))+(42-17)) = 50
```

```
    # $a0 : pointer to expr.
eval:
    lb      $t0, 0($a0)
    bne     $t0, $0, dobranch

# Leaf: return its value
    lw      $v0, 4($a0)
    jr      $ra

dobranch:
# Save $ra, $s0, and $s1 on stack
    addiu   $sp, $sp, -16
    sw      $ra, 0($sp)
    sw      $s0, 4($sp)
    sw      $s1, 8($sp)

# Eval left tree (to $s0)
    move    $s1, $a0
    lw      $a0, 4($a0)
    jal     eval
    move    $s0, $v0

# Eval right tree (to $v0)
    lw      $a0, 8($s1)
    jal     eval
```

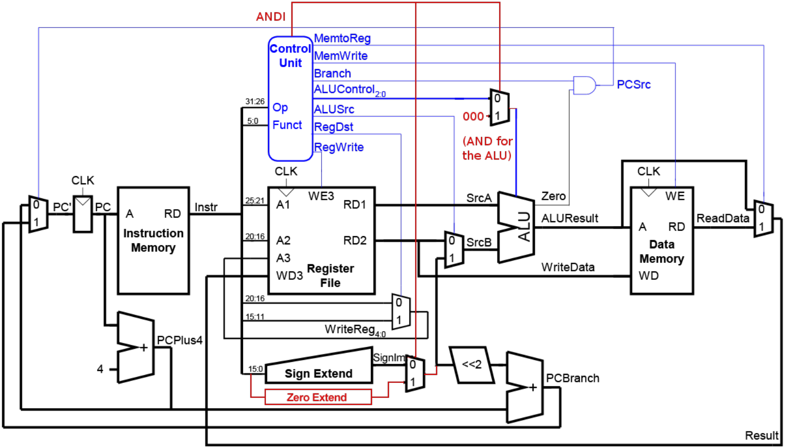
```
    lb      $t0, 0($s1)
    li      $t1, '+'
    bne     $t0, $t1, L1
# Operator was +: add
    addu    $v0, $s0, $v0
    b       evalexit

L1: li      $t1, '-'
    bne     $t0, $t1, L2
# Operator was -: subtract
    subu    $v0, $s0, $v0
    b       evalexit

L2: li      $t1, '*'
    bne     $t0, $t1, evalexit
# Operator was *: multiply
    mul     $v0, $s0, $v0

evalexit:
# Restore $ra, $s0, and $s1
    lw      $ra, 0($sp)
    lw      $s0, 4($sp)
    lw      $s1, 8($sp)
    addiu   $sp, $sp, 16
    jr $ra
```

3. (25 pts.) Extend the single-cycle MIPS processor to support the `andi` instruction (i-type, OP=001100).



Inst.	OP	RegWrite	RegDst	ALUSrc	Branch	MemWrite	MemToReg	ALUOp	ANDI
R-type	000000	1	1	0	0	0	0	1-	0
lw	100011	1	0	1	0	0	1	00	0
sw	101011	0	-	1	0	1	-	00	0
beq	000100	0	-	0	1	0	-	01	0
andi	001100	1	0	1	0	0	0	--	1

4. (10 pts.) Assuming the following dynamic instruction frequency for a program running on the single-cycle MIPS processor

`addu`	25%
`addi`	25%
`beq`	15%
`lw`	20%
`sw`	15%

(a) (5 pts.) In what fraction of all cycles is the data memory accessed (either read or written)?

Only for loads and stores, so 20% (lw) + 15% (sw) = 35%.

(b) (5 pts.) In what fraction of cycles is the sign extend circuit used?

addi uses it for the immediate operand
beq uses it to compute the PC-relative address
lw uses it to compute the offset address
sw uses it to compute the offset address
So, 25% + 15% + 20% + 15% = 75%.

5. (15 pts.) For each of the caches listed below, show how a 32-bit addresses breaks into *tag*, *set index*, and *byte offset* fields.

Cache A: 8192B, 4-way set-associative, 16B lines

64B per set, so 128 sets in cache

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

tag (21 bits) | set index (7 bits) | byte offset (4 bits)

Cache B: 4096B, direct-mapped, 8B lines

8B per set, so 512 sets in cache

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

tag (20 bits) | set index (9 bits) | byte offset (3 bits)