CSC258 - MIPS Assembly

JoJo – EZ4

Main Topics

- Branch (if/else)
- Jump and Repeat (loops)
- Array(list) and Struct
- Stack and Return

Branch Instruction

- beq – branch out **if $s == $t**
- bgtz – branch out **if $s > 0**
- blez – branch out **if $s <=0**
- bne – branch out **if $s!=$t**

Instruction	Opcode/Function	Syntax	Operation
beq	000100	$s, $t, label	if ($s == $t) pc += i << 2
bgtz	000111	$s, label	if ($s > 0) pc += i << 2
blez	000110	$s, label	if ($s <= 0) pc += i << 2
bne	000101	$s, $t, label	if ($s != $t) pc += i << 2

```
if (i == j)
        i++;
else
        j--;

j = j + i;
```

```
# $t1 = i, $t2 = j

main :   bne $t1, $t2, ELSE // branch if $t1 != $t2
         addi $t1, $t1, 1
         j END
ELSE :   addi $t2, $t2, -1
END  :   add $t2, $t2, $t1
```

If We change BNE to BEQ

```
# $t1 = i, $t2 = j

main :    bne $t1, $t2, ELSE // branch if $t1 != $t2
          addi $t1, $t1, 1
          j END
ELSE :    addi $t2, $t2, -1
END  :    add $t2, $t2, $t1
```

```
# $t1 = i, $t2 = j

main :    beq $t1, $t2, IF // branch if $t1 == $t2
          addi $t2, $t2, -1
          j END
IF :      addi $t1, $t1, 1
END  :    add $t2, $t2, $t1
```

Loops(While Structure)

- While (<**cond**>) {<**body**>}
  - Similar to if/else structure
  - Can simply use beq/bne··· to implement

- Example:

```
while (i<100)
{
        i++;
}
```

```
# $t0 = i, $t1= n
main:   add $t0, $zero, $zero    #i = 0
        addi $t1, $zero, 100     # n = 100

START:  beq $t0, $t1, END        # if i == n, END
        addi $t0, $t0, 1         # i++
        j START
END:
```

Loops(For Structure)

- for (<init>; <cond>; <update>) {<for body>}

- Assembly equivalent:
- main: <init>
- START: if (!<cond>) branch to END

```
            <for body>
UPDATE:     <update>
            j START
END:
```

Memory Access

- Play with registers and imm values – DONE
- How to compute value associate memory?
- How to return values?
- How to store list and struct?

Instruction	Opcode/Function	Syntax	Operation
lb	100000	$t, i ($s)	$t = SE (MEM [$s + i]:1)
lbu	100100	$t, i ($s)	$t = ZE (MEM [$s + i]:1)
lh	100001	$t, i ($s)	$t = SE (MEM [$s + i]:2)
lhu	100101	$t, i ($s)	$t = ZE (MEM [$s + i]:2)
lw	100011	$t, i ($s)	$t = MEM [$s + i]:4
sb	101000	$t, i ($s)	MEM [$s + i]:1 = LB ($t)
sh	101001	$t, i ($s)	MEM [$s + i]:2 = LH ($t)
sw	101011	$t, i ($s)	MEM [$s + i]:4 = $t

'w' : word 4 byte(32bit)
'h': half word 2byte(16bit)
'b': byte (8 bit)
LB: lowest byte
LH lowest half word

Memory Access

- **lh $t0, 12($s0)** syntax：lh reg offset(mem)
- #Load a half-word (2 bytes) starting from MEM($s0 + 12), sign-extend it to 4 bytes, and store in $t0
- **sb $t0, 12($s0)** syntax：sb reg offset(mem)
- Take the lowest byte of the word stored in $t0, store it to memory starting from address $s0 + 12

- * The offset is very useful when we dealing with array, list, struct etc.

Arrays and Struct

- At beginning of program, create labels for memory locations that are used to store values.
- Always in form: **label type value**

- Example

```
.data
array1:     .word   3, 7, 5, 42        #int type = 4 byte = 1 word
array2:     .byte   'e', 'z'           #char type  = 1 byte
array3:     .space  40                 #40 byte of space
```

Array (List)

- The address of the first element of the array is used to store and access the elements of the array.

- To access an element of the array, get the address of that element by adding an offset distance to the address of the first element.
  - **offset = array index * the size of a single element**

- Arrays are stored in **memory**. For examples, fetch the array values and store them in registers. Operate on them, then store them back into memory.

```
int A[100], B[100];
for (i=0; i<100; i++) {
   A[i] = B[i] + 1;
}
```

```
.data
A:          .space    400
B:          .space    400

.text
main:       add $t0, $zero, $zero
            addi $t1, $zero, 400
            la $t8, A
            la $t9, B

loop:       add $t4, $t8, $t0
            add $t3, $t9, $t0
            lw $s4, 0($t3)
            addi $t6, $s4, 1
            sw $t6, 0($t4)
            addi $t0, $t0, 4
            bne $t0, $t1, loop
end:
```

Initialization:

- Allocate space
- Initial value i=0 (offset), put into a register
- Put value size (400) in register
- Put addresses of A, B into register

The loop:

- Put addrs of A[i] and B[i] into registers (addr(A)+offset).
- Load B[i] from mem, then +1, keep result in a register
- Store result into mem A[i]
- Update i++
- Check loop condition and jump

Struct

```
struct food
{
        int price;
        char label;
        int num;
}


struct food x;
x.price = 5;
x.num = 3;
x.label = 'B'
```

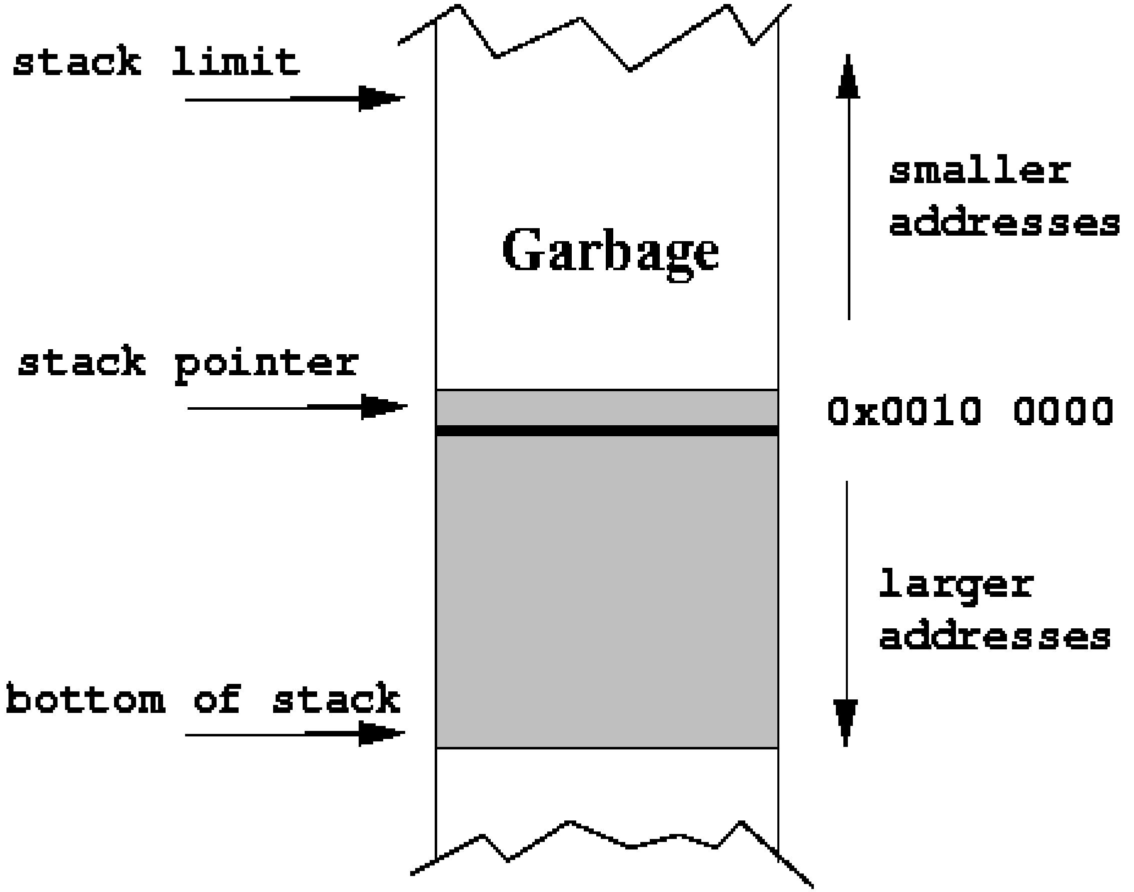
```
            .data

struct1:    .space      9

            .text

main:       la      $t0, struct1   #load address
            addi    $t1, $zero, 5
            sw      $t1, 0($t0)

            addi    $t1, $zero, 'B'   #store ascii 66
            sb      $t1, 4($t0)

            addi    $t1, $zero, 3
            sw      $t1, 5($t0)
```

Function call (return)

- **Function arguments** stores in Memory (in a stack structure)

- **stack grows backwards**, i.e., when stack *pointer (top) decreases, stack becomes bigger; when stack pointer increase, stack becomes smaller.*

- Store in stack cause there is not enough registers



Access Stack

- The address of the "top" of the stack is stored in this register -- **$sp**
- **Stack Pointer**

- PUSH value in $t0 into stack

```
addi $sp, $sp, -4        # make more space
sw   $t0, 0($sp)         # push word onto the stack
```

- POP value from stack and store in $t0

```
lw $t0, 0($sp)           # pop from stack
addi $sp, $sp, 4         # make stack smaller
```

Example

```c
int sign (int i)
{
	if (i > 0)
		return 1;
	else if (i < 0)
		return -1;
	else
		return 0;
}
```

```asm
.text
sign:	lw $t0, 0 ($sp)
	addi $sp, $sp, 4

	bgtz $t0, gt
	beq  $t0, $zero, eq
	addi $t1, $zero, -1
	j end

gt:	addi $t1, $zero, 1
	j end

eq:	add $t1, $zero, $zero

end:	addi $sp, $sp, -4
	sw $t1, 0($sp)

	jr $ra
```

考题重温

- 形式1：给出命令描述， 求MIPS命令 （3‘/per）
- 形式2： 给出Assembly Program， 求descriptive comments, overall one-sentence description and de-bug one line (if possible) (10-12’ total)
- 形式3：给出simple C – program， 求assembly