CS 61C Spring 2010
Section 114/8

Week 4 – MIPS! (revised)

TA: Michael Greenbaum
cs61c-tf@imail.eecs.berkeley.edu

MIPS cheat sheet (including instructions you hadn't learned at the time)

Instruction	Syntax	Example
`add`	`add  dest, src0, src1`	`add  $s0, $s1, $s2`
`sub`	`sub  dest, src0, src1`	`sub  $s0, $s1, $s2`
`addi`	`addi dest, src0, immediate`	`addi $s0, $s1, 12`
`sll / srl`	`sll  dest, src,  immediate`	`sll   $t0, 4($s0)`
`lw / lb`	`lw   dest, offset(base addr)`	`lw   $t0, 4($s0)`
`sw / sb`	`sw   src,  offset(base addr)`	`sw   $t0, 4($s0)`
`bne`	`bne  src0, src1, branchAddr`	`bne  $t0, $t1, notEq`
`beq`	`beq  src0, src1, branchAddr`	`bne  $t0, $t1, Eq`
`j`	`j    jumpAddr`	`j    jumpWhenDone`
`jr`	`jr   reg`	`jr   $ra`

C	MIPS

`// $s0 -> a (use $s0 for a), $s1 -> b`
`// $s2 -> c, $s3 -> z`
 	
`int a=4, b=5, c=6, z;`
`z = a+b+c+10;`
	`addi $s0, $0, 4`
	`addi $s1, $0, 5`
	`addi $s2, $0, 6`
	`add $s3, $s0, $s1`
	`add $s3, $s3, $s2`
	`addi $s3, $s3, 10`

`// $s0 -> int *p = (int *)malloc`
`//                  (3*sizeof(int));`
`// $s1 -> a`
`p[0] = 0;`
`int a = 2;`
`p[1] = a;`
`p[a] = a;`
	`sw $0, 0($s0)`
	`addiu $s1, $0, 2`
	`sw $s1, 4($s0)`
	`sll $t0, $s1, 2 #same as <<`
	`addu $t1, $t0, $s0`
	`sw $s1, 0($t1)`

`// $s0 -> a, $s1 -> b`
`int a = 5, b = 10;`
`if (a + a == b) {`
`    a = 0;`
`} else {`
`    b = a - 1;`
`}`
	`addiu $s0, $0, 5`
	`addiu $s1, $0, 10`
	`add $t0, $s0, $s0`
	`bne $t0, $s1, else`
	`add $s0, $0, $0`
	`j exit`
	`else: addiu $s1, $s0, -1`
	`exit: # done!`

`/*What does this do?`
`  (Not C, in English) */`
Returns 2^30, or 2^N where N is the immediate on line 3
	`      addi $s0, $0,  0`
	`      addi $s1, $0,  1`
	`      addi $t0, $0,  30`
	`loop: beq  $s0, $t0, done`
	`      add  $s1, $s1, $s1`
	`      addi $s0, $s0, 1`
	`      j    loop`
	`done: # done!`

```
// Strcpy:
// $s1 -> char s1[] = "Hello!";
// $s2 -> char *s2 =
//          malloc(sizeof(char)*7);
int i=0;
do{
    s2[i] = s1[i];
    i++;
} while(s1[i]!='\0')
```

Doesn't actually work since it doesn't copy over the null terminator…

```
addi $t0, $0, 0
loop: add $t1, $s1, $t0
add $t2, $s2, $t0
lb $t3, 0($t1)
sb $t3, 0($t2)
addi $t0, $t0, 1
addi $t1, $t1, 1
lb $t4, 0($t1)
beq $t4, $0, done
j loop
done: # done!
```

```
// Nth_Fibonacci(N):
// $s0 -> N, $s1 -> fib
// $t0 -> i, $t1 -> j
if(N==0) return 0;
else if(N==1) return 1;
N-=2;
int fib=1, i=1, j=1;
while(N!=0){
    fib = i+j;
    j = i;
    i = fib;
    N--;
}
return fib;
```

```
beq $s0, $0, returnZero
addi $t0, $0, 1
beq $s0, $t0, returnOne
subi $s0, $s0, 2
addi $s1, $0, 1
addi $t0, $0, 1
addi $t1, $0, 1
loop: beq $s0, $0, returnFib
add $s1, $t0, $t1
addi $t0, $t1, 0
addi $t1, $s1, 0
subi $s0, $s0, 1
j loop
returnZero: addi $v0 $0 0
j done
returnOne: addi $v0 $0 1
j done
returnFib: add $v0 $0 $s1
done: jr $ra
```

Iterative fibonacci.

```
      #0x100-0x104 valid addresses
      add  $s0, $0,  $0
      addi $s1, $0,  10
      add  $s2, $0,  $0
      addi $s3, $0,  0x100
loop: beq  $s0, $s1, done
      addi $t0, $s0, 4
      lw   $t0, 0($s3)
      lw   $t1, 4($s3)
      sw   $t1, 0($s3)
      add  $s2, $s2, $t0
      add  $s2, $s2, $t1
      sw   $s2, 4($s0)
      addi $s0, $s0, 1
      j    add
add:  addi $s0, $s0, 1
      j    loop
done: # done!
```

Fill in the blanks in the MIPS code. Also add jump labels in appropriate places

```
// 0x100 -> &a, 0x200 -> &b
// $s0 -> i

int a[4], b[4];
```

```
         addi $s0, $0,  4
         beq  $s0, $0,  done
         add  $t0, $0,  4
         addi $t1, $0,  $s0
         addi $t2, $0,  0
do_mult: beq  $t0, $0,  copy
```

```
int i;
for (i = 4; i != 0; i--) {
    b[i] = a[i];
}
```

```
      add  $t2, $t2, $t1
      sub  $t0, $t0, 1
      j    do_mult
copy: lw   $t0, 0x100($t1)
      sw   $t0, 0x200($t1)
      subi $s0, $s0, 1
      j    loop
done: # done!
```

Editor's note: I spent half an hour trying to get rid of this extra space. Tables in word are terrible.

Solutions adopted from Long Wei.