EECE 321: Computer Organization

Mohammad M. Mansour
Dept. of Electrical and Compute Engineering
American University of Beirut

Practice Problems

Arrays

Problem 1

- Compile the following code from C into MIPS
  - First do it assuming you can use pseudo-instructions
  - Then, replace pseudo-instructions with corresponding true MIPS instructions

```
# i:$s0, j:$s1, A:$s2
for(i=0; i<10; i++)
    for(j=0; j<=i; j++)
          A[i+16*j] = 13*A[i+j];
```

Problem 2

- Compiling arrays whose indices involve another memory reference
- Ex: Compile A[B[4]]
  - cannot be written in MIPS as [4(16($s7))]($s6)
- First need to break it into pieces
  - Compile j = B[4]
  - Then compile A[i]
- Ex: Compile A[B[j]] #$s0:A[], $s1:B[], $s2:j
  - sll $t0, $s2,2 # 4*j
  - add $t0,$t0,$s1 # 4*j + base of B[]
  - Lw $t0, 0($t0) # t0 = B[j]; note that B[j] is an index
  - sll $t0,$t0,2 # multiply index 'B[j]' by 4: 4*B[j]
  - add $t0,$t0,$s0 # 4*B[j] + base of A[]
  - lw $t0,0($t0) # load
- Note: try to minimize as much as possible the number of temporary registers used
  - **Above we used only $t0**

Problem 3: Clearing the Contents of an Array

- Compile the following C-function into MIPS:

```
clear (int array[], int size){
  int i;
  for(i=0;i<size;i++)
      array[i]=0;
}
```

- Choose your own variable-register mapping.

Problem 3 (cont'd): Clearing the Contents of an Array

```
clear (int array[], int size){
  int i;
  for(i=0;i<size;i++)
      array[i]=0;
}
```

```
# array:$a0, size:$a1, i:$t0

        li   $t0, 0
FOR:    slt  $t1, $t0, $a1
        beq  $t1, $0, EXIT
        sll  $t1, $t0, 2    # 4*i
        add  $t1, $t1, $a0
        sw   $0, 0($t1)
        addi $t0, $t0, 1
        j    FOR
EXIT:   jr   $ra
```

Arithmetic/Overflow

Problem 4

- Suppose the instructions slt, sltu, slti, sltiu were removed from the MIPS instruction set. Show how to perform slt $s0,$s1,$s2 using the modified instruction set in which slt is not available (no pseudo-instructions allowed). Beware to account for overflow.
- **Solution**:

```
if ($s0<0) and ($s1>0) then
        $t0:=1
else if ($s0>0) and ($s1<0) then
        $t0:=0
else
        $t1:=$s0-$s1

if ($t1<0) then
        $t0:=1
else
        $t0:=0
```

Dissassembly

Problem 5

- Consider the following MIPS code:

```
        addi $t1 , $0 , 100
  loop: lw $s1 , 0($s0)
        add $s2 , $s2 , $s1
        addi $s0 , $s0 , 4
        subi $t1 , $ t1 , 1
        bne $t1 , $0 , loop
```

- Disassemble this code into C, and try to optimize it
- First trial:

```
// n : $t1; i: $s0; x : $s1; y:$s2
n = 100;
while (n != 0){
   x = A[i];
   y = y + x;
   n-- ;
}
```

→

```
i =0;
while ( i < 100){
   y = y + A[i++];
}
```

Branches

Problem 6

- How to branch to a location that is further than what can be represented in the immediate field in a branch instruction?
  - Solution: Branch to a "jr" instruction
- Ex: if $s1=$s2 jump to L, but L is very far away

```
      beq $s1 , $s2 , L1
      j L2
L1:   li $t0 , L     # here L is a 32-bit value that represents the 'far' address
      jr $t0
L2:
```

- What is the disadvantage of this approach? What needs to be changed if this code is moved in memory?

Problem 7

- Suppose that MIPS does not have conditional branch instructions (beq, bne, …). Can you write a sequence of MIPS instructions to implement a conditional branch instruction?

Problem 8

- Translate the following switch statement in C into MIPS. Try to use as few registers as possible. Assume the following mapping: \$s1:x, \$s2:y, \$s3:z, \$s0:a. Add appropriate comments to your MIPS code.

```
switch(a){
      case 1: x = y + z;
      case 2: x = y - z;
      case -1: x = x - z;
      default: x = z + z;
}
```

Solution:

```
          addi $t0, $0, 1          # case a=1
          addi $t1, $0, 2          # case a=2
          addi $t2, $0, -1         # case a=-1

          bne  $s0, $t0, L2        # handle case 1
          add  $s1, $s2, $s3       #
          j    EXIT

L2:       bne  $s0, $t1, L3        # handle case 2
          sub  $s1, $s2, $s3
          j    EXIT

L3:       bne  $s0, $t2, D         # handle case -1
          sub  $s1, $s1, $s3
          j    EXIT

D:        add  $s1, $s3, $s3
EXIT:
```

Problem 9

- Implement the instruction max $t1,$t2,$t3 in MIPS, which returns the maximum of $t2 and $t3 in $t1, without using any conditional branch instructions (beq, bne). Also, you are not allowed to use pseudo-instructions.
- <u>Solution:</u>

```
        add $t1,$t2,$0          # MAX is t2
        jal MAX                 # $ra = address MAX

MAX:    slt $t0,$t3,$t2         # t0=1 if t2 is MAX, exit
        sll $t0,$t0,2           # t0 is 4
        addi $ra,$ra,20         # adjust jump offset
        add $ra,$ra,$t0
        jr $ra

T3MAX:  add $t1,$0,$t3          # t3 is MAX

T2MAX:  ……
```

Functions

Problem 10: Sorting Function

- Compile the following C-function into MIPS:

```
sort (int v[], int n){
   int i,j;
   for(i=0;i<n;i++){
        for(j=i-1;j>=0 && v[j]>v[j+1]; j--){
               swap(v,j);
        }
   }
}
```

```
swap (int v[], int k){
   int temp;
   temp=v[k];
   v[k]=v[k+1];
   v[k+1]=temp;
}
```

- Steps:
  - Allocate registers to program variables.
  - Produce code for the body of the function.
  - Preserve registers across the procedure invocation.
- Compiling swap: `#v:$a0, k:$a1`

```
swap:   sll  $t1,$a1,2          # $t1=k*4
        add  $t1,$a0,$t1        # $t1=v+4*k, $t1 has address of v[k]
        lw   $t0,0($t1)         # temp = v[k]
        lw   $t2,4($t1)         # $t2 = v[k+1]
        sw   $t2,0($t1)         # v[k] = $t2
        sw   $t0,4($t1)         # v[k+1] = temp
        jr   $ra                # return to caller
```

Problem 10 (cont'd): Compiling sort()

- First for loop: **#i:s0, j:$s1, n:$a1**

```
sort (int v[], int n){
   int i,j;
   for(i=0;i<n;i++){
         for(j=i-1;j>=0 && v[j]>v[j+1]; j--){
                  swap(v,j);
         }
   }
}
```

```
         add  $s0,$zero,$zero              # i=0
FOR1:    slt  $t0,$s0,$a1                  # if i>=n
         beq  $t0,$zero,EXIT1              # exit loop
         . . .
            (body of first loop)
         . . .
         addi $s0,$s0,1                    # i++
         j    FOR1
EXIT1:
```

Problem 10 (cont'd): Compiling sort()

```
sort (int v[], int n){
   int i,j;
   for(i=0;i<n;i++){
         for(j=i-1;j>=0 && v[j]>v[j+1]; j--){
                  swap(v,j);
         }
   }
}
```

- Second for loop: **#i:s0, j:$s1**

```
         addi  $s1,$s0,-1                   # j=i-1
FOR2:    slti  $t0,$s1,0                    # $t0=1 if j<0
         bne   $t0,$zero,EXIT2              # exit loop if j<0
         sll   $t1,$t1,2                    # $t1 = 4*j
         add   $t2,$a0,$t1                  # $t2 = v + 4*j
         lw    $t3,0($t2)                   # $t3 = v[j]
         lw    $t4,4($t2)                   # $t4 = v[j+1]
         slt   $t0,$t4,$t3                  # $t0=0 if $t4 >= $t3
         beq   $t0,$zero,EXIT2              # goto EXIT2 if $t4 >= $t3
         . . .
            (body of second loop)
         . . .
         addi $s1,$s1,1                     # j--
         j    FOR2
EXIT2:
```

Problem 10 (cont'd): Compiling sort()

- Preserving registers:
  - Need to save $a0, $a1. Can do so on stack, but better copy them to unused registers $s2,$s3.

```
add  $s2,$a0,$zero     # move $a0 into $s2
add  $s3,$a1,$zero     # move $a1 into $s3
```

- **Pass parameters to swap as:**

```
add  $a0,$s2,$zero     # first swap parameter is v
add  $a1,$s1,$zero     # second swap parameter is j
```

- **Since sort uses saved registers, it must save them on stack in addition to $ra:**

```
addi  $sp,$sp,-20              # make room for 5 registers
sw    $ra,16($sp)              # save $ra
sw    $s3,12($sp)              # save $s3
sw    $s2, 8($sp)              # save $s2
sw    $s1, 4($sp)              # save $s1
sw    $s0, 0($sp)              # save $s0
```

- **Epilogue simply pops the stack.**
- **See the overall code in your textbook.**