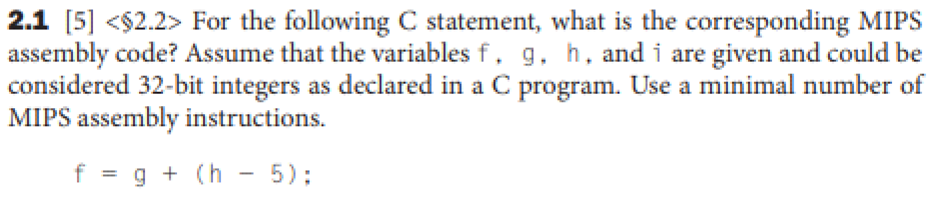
ECE4300 HW 2 Kenneth Bach

Sunday, October 13, 2024 3:33 PM

2.1, 2.2, 2.3, 2.4, 2.5, 2.6, 2.7, 2.9, 2.10, 2.12, 2.18, 2.21, 2.22, 2.26, 2.29, 2.38, 2.47

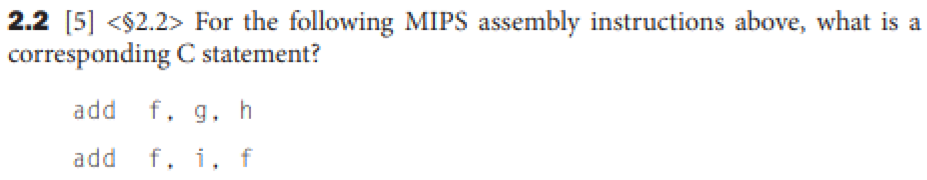


2.1 [5] <§2.2> For the following C statement, what is the corresponding MIPS assembly code? Assume that the variables f, g, h, and i are given and could be considered 32-bit integers as declared in a C program. Use a minimal number of MIPS assembly instructions.

```
f = g + (h − 5);
```

Using temp registers: f in $t0, g in $t1, h in $t2

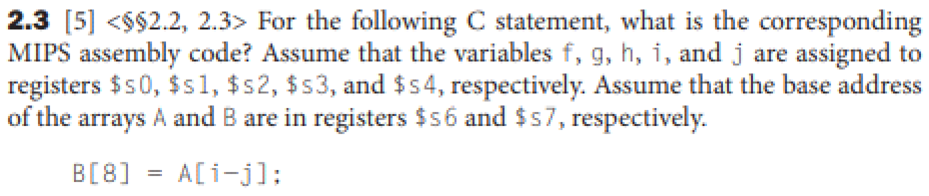
addi $t3, $t2, -5	# $t3 = h-5
Add $t0, $t1, $t3	# f = g+(h-5)



2.2 [5] <§2.2> For the following MIPS assembly instructions above, what is a corresponding C statement?

```
add  f, g, h
add  f, i, f
```

F = g+h	
F = i+f	



2.3 [5] <§§2.2, 2.3> For the following C statement, what is the corresponding MIPS assembly code? Assume that the variables f, g, h, i, and j are assigned to registers $s0, $s1, $s2, $s3, and $s4, respectively. Assume that the base address of the arrays A and B are in registers $s6 and $s7, respectively.

```
B[8] = A[i−j];
```

sub $t0, $s3, $s4	# $t0 = i - j
sll $t1, $t0, 2	# $t1 = (i - j) * 4
add $t2, $s6, $t1	# $t2 = Address of A[i - j]
lw $t3, 0($t2)	# Load A[i - j] into $t3
sw $t3, 32($s7)	# Store into B[8] (8 * 4 = 32)

2.4 [5] <§§2.2, 2.3> For the MIPS assembly instructions below, what is the corresponding C statement? Assume that the variables f, g, h, i, and j are assigned to registers $s0, $s1, $s2, $s3, and $s4, respectively. Assume that the base address of the arrays A and B are in registers $s6 and $s7, respectively.

```
sll  $t0, $s0, 2     # $t0 = f * 4
add  $t0, $s6, $t0   # $t0 = &A[f]
sll  $t1, $s1, 2     # $t1 = g * 4
add  $t1, $s7, $t1   # $t1 = &B[g]
lw   $s0, 0($t0)     # f = A[f]
addi $t2, $t0, 4
lw   $t0, 0($t2)
add  $t0, $t0, $s0
sw   $t0, 0($t1)
```

f = A[f];	
B[g] = A[f+1] +f	

#1	F * 4 for byte offset
#2	Introduce A ($s6) to $t0
#3	G *4 for byte offset
#4	Introduce B ($s7) to $t1
#5	Load A[f]
#6	Add 4 to &A[f] ($t0) setting up A[f+1] ($t2)
#7	Load ($t2) into ($t0)
#8	Add f ($s0) to A[f+1] ($t0)
#9	Store the sum in B[g] ($t1)

2.5 [5] <§§2.2, 2.3> For the MIPS assembly instructions in Exercise 2.4, rewrite the assembly code to minimize the number if MIPS instructions (if possible) needed to carry out the same function.

sll $t0, $s0, 2	# $t0 = f * 4
add $t0, $s6, $t0	# $t0 = &A[f]
sll $t1, $s1, 2	# $t1 = g * 4
add $t1, $s7, $t1	# $t1 = &B[g]
lw $t2, 0($t0)	# $t2 = A[f]
lw $t3, 4($t0)	# $t3 = A[f + 1]
add $t2, $t2, $t3	# $t2 = A[f] + A[f + 1]
sw $t2, 0($t1)	# B[g] = A[f] + A[f + 1]

2.6 The table below shows 32-bit values of an array stored in memory.

Address	Data
24	2
38	4
32	3
36	6
40	1

2.6.1 [5] <§§2.2, 2.3> For the memory locations in the table above, write C code to sort the data from lowest to highest, placing the lowest value in the smallest memory location shown in the figure. Assume that the data shown represents the C variable called `Array`, which is an array of type `int`, and that the first number in the array shown is the first element in the array. Assume that this particular machine is a byte-addressable machine and a word consists of four bytes.

2.6.2 [5] <§§2.2, 2.3> For the memory locations in the table above, write MIPS code to sort the data from lowest to highest, placing the lowest value in the smallest memory location. Use a minimum number of MIPS instructions. Assume the base address of `Array` is stored in register `$s6`.

lw $t0, 0($s6)	# Load value at address 24 (Array[0])
lw $t1, 14($s6)	# Load value at address 38 (Array[1])
lw $t2, 8($s6)	# Load value at address 32 (Array[2])
lw $t3, 12($s6)	# Load value at address 36 (Array[3])
lw $t4, 16($s6)	# Load value at address 40 (Array[4])
li $t5, 0	# $t5 is the swap flag, initialize to 0
bge $t0, $t2, swap_0_2	
nop	
bge $t3, $t4, swap	

2.7 [5] <§2.3> Show how the value `0xabcdef12` would be arranged in memory of a little-endian and a big-endian machine. Assume the data is stored starting at address 0.

0x12 0xEF 0xCD 0xAB	Little-endian
0xAB 0xCD 0xEF 0x12	Big-endian

2.9 [5] <§§2.2, 2.3> Translate the following C code to MIPS. Assume that the variables f, g, h, i, and j are assigned to registers $s0, $s1, $s2, $s3, and $s4, respectively. Assume that the base address of the arrays A and B are in registers $s6 and $s7, respectively. Assume that the elements of the arrays A and B are 4-byte words:

```
B[8] = A[i] + A[j];
```

sll $t0, $s3, 2	# $t0 = i * 4 (shift left logical by 2)
add $t0, $s6, $t0	# $t0 = &A[i] (base address of A + i * 4)
lw $t1, 0($t0)	# $t1 = A[i]
sll $t2, $s4, 2	# $t2 = j * 4 (shift left logical by 2)
add $t2, $s6, $t2	# $t2 = &A[j] (base address of A + j * 4)
lw $t3, 0($t2)	# $t3 = A[j]
add $t4, $t1, $t3	# $t4 = A[i] + A[j]
li $t5, 32	# $t5 = 8 * 4 (since B[8] is 32 bytes from base)
add $t5, $s7, $t5	# $t5 = &B[8] (base address of B + 32)
sw $t4, 0($t5)	# B[8] = A[i] + A[j]

2.10 [5] <§§2.2, 2.3> Translate the following MIPS code to C. Assume that the variables f, g, h, i, and j are assigned to registers $s0, $s1, $s2, $s3, and $s4, respectively. Assume that the base address of the arrays A and B are in registers $s6 and $s7, respectively.

```
addi $t0, $s6, 4
add  $t1, $s6, $0
sw   $t1, 0($t0)
lw   $t0, 0($t0)
add  $s0, $t1, $t0
```

2.12 Assume that registers `$s0` and `$s1` hold the values `0x80000000` and `0xD0000000`, respectively.

2.12.1 [5] <§2.4> What is the value of `$t0` for the following assembly code?

```
add $t0, $s0, $s1
```

2.12.2 [5] <§2.4> Is the result in `$t0` the desired result, or has there been overflow?

2.12.3 [5] <§2.4> For the contents of registers `$s0` and `$s1` as specified above, what is the value of `$t0` for the following assembly code?

```
sub $t0, $s0, $s1
```

2.12.4 [5] <§2.4> Is the result in `$t0` the desired result, or has there been overflow?

2.12.5 [5] <§2.4> For the contents of registers `$s0` and `$s1` as specified above, what is the value of `$t0` for the following assembly code?

```
add $t0, $s0, $s1
add $t0, $t0, $s0
```

2.12.6 [5] <§2.4> Is the result in `$t0` the desired result, or has there been overflow?

2.18 Assume that we would like to expand the MIPS register file to 128 registers and expand the instruction set to contain four times as many instructions.

2.18.1 [5] <§2.5> How this would this affect the size of each of the bit fields in the R-type instructions?

2.18.2 [5] <§2.5> How this would this affect the size of each of the bit fields in the I-type instructions?

2.18.3 [5] <§§2.5, 2.10> How could each of the two proposed changes decrease the size of an MIPS assembly program? On the other hand, how could the proposed change increase the size of an MIPS assembly program?

2.21 [5] <§2.6> Provide a minimal set of MIPS instructions that may be used to implement the following pseudoinstruction:

```
    not $t1, $t2      // bit-wise invert
```

nor $t1, $t2, $zero # $t1 = ~$t2	

2.22 [5] <§2.6> For the following C statement, write a minimal sequence of MIPS assembly instructions that does the identical operation. Assume `$t1 = A`, `$t2 = B`, and `$s1` is the base address of C.

```
A = C[0] << 4;
```

lw $t0, 0($s1)	# Load C[0] into $t0
sll $t1, $t0, 4	# Shift the value in $t0 left by 4 bits, store in $t1

2.26 Consider the following MIPS loop:

```
LOOP: slt  $t2, $0,  $t1
      beq  $t2, $0,  DONE
      subi $t1, $t1, 1
      addi $s2, $s2, 2
      j    LOOP
DONE:
```

2.26.1 [5] <§2.7> Assume that the register `$t1` is initialized to the value 10. What is the value in register `$s2` assuming `$s2` is initially zero?

2.26.2 [5] <§2.7> For each of the loops above, write the equivalent C code routine. Assume that the registers `$s1`, `$s2`, `$t1`, and `$t2` are integers `A`, `B`, `i`, and temp, respectively.

2.26.3 [5] <§2.7> For the loops written in MIPS assembly above, assume that the register `$t1` is initialized to the value N. How many MIPS instructions are executed?

2.29 [5] <§2.7> Translate the following loop into C. Assume that the C-level integer `i` is held in register `$t1`, `$s2` holds the C-level integer called `result`, and `$s0` holds the base address of the integer `MemArray`.

```
       addi $t1, $0, $0
LOOP:  lw   $s1, 0($s0)
       add  $s2, $s2, $s1
       addi $s0, $s0, 4

      addi $t1, $t1, 1
      slti $t2, $t1, 100
      bne  $t2, $s0, LOOP
```

2.38 [5] <§2.9> Consider the following code:

```
lbu $t0, 0($t1)
sw  $t0, 0($t2)
```

Assume that the register `$t1` contains the address `0x1000 0000` and the register `$t2` contains the address `0x1000 0010`. Note the MIPS architecture utilizes big-endian addressing. Assume that the data (in hexadecimal) at address `0x1000 0000` is: `0x11223344`. What value is stored at the address pointed to by register `$t2`?

2.47 Assume that for a given program 70% of the executed instructions are arithmetic, 10% are load/store, and 20% are branch.

2.47.1 [5] <§2.19> Given this instruction mix and the assumption that an arithmetic instruction requires 2 cycles, a load/store instruction takes 6 cycles, and a branch instruction takes 3 cycles, find the average CPI.

2.47.2 [5] <§2.19> For a 25% improvement in performance, how many cycles, on average, may an arithmetic instruction take if load/store and branch instructions are not improved at all?

2.47.3 [5] <§2.19> For a 50% improvement in performance, how many cycles, on average, may an arithmetic instruction take if load/store and branch instructions are not improved at all?