EECS 645 Homework 6: MIPS ISA

Seamus Herrod

KU ID: 2989137

2.24 Suppose the program counter (PC) is set to 0x2000 0000. Is it possible to use the jump (j) MIPS assembly instruction to set the PC to the address as 0x4000 0000? Is it possible to use the branch-on-equal (beq) MIPS assembly instruction to set the PC to this same address?

Solution:

0x2000 0000 → 0010 0000 0000 0000 0000 0000 0000 0000

j Lbl → [6-bit opCode | 26-bit address]

jump instruction is capable of changing the 26 LSBs of PC (because the address in the j instr is only 26 bits)

- if order for the PC to be set from 0x2000 0000 → 0x4000 0000 the four MSBs of PC would need to be acccessed

beq intr → [6-bit OpCode | 5-bit rs | 5-bit rt | 16-bit offset]

beq can only be used to change the value of the least significant 2 bytes of the address

- 0x0000 0000 → 0x0000 FFFF

2.25 The following instruction is not included in the MIPS instruction set:

```
rpt $t2, loop if(R[rs]>0) R[rs]=R[rs]
-1, PC=PC+4+BranchAddr
```

2.25.1 If this instruction were to be implemented in the MIPS instruction set, what is the most appropriate instruction format?

Solution:

R-type instructions are of the form:

```
[6-bit opcode | 5-bit rs | 5-bit rt | 5-bit rd | 5-bit shamt | 6-bit funct ]
```

- This doesn't seem like a good format for rpt because there would be 3 unused fields and the BranchAddressed would be restricted to only 6-bits
- J-type instructions have a 26 bit branch address field but no bit-field for the rs required by rpt
- This leaves I-type as a suitable format for the rpt instruction. Only one field would be unused (the rt bit-field) but would leave a 16-bit branch address field

I-format

2.25.2 What is the shortest sequence of MIPS instructions that performs the same operation?

Solution:

```
rpt: slt $t0, $0, $t2
     beq $t0, $0, EXIT
     addi $t2, $t2, -1
     j loop
EXIT:
```

2.26 Consider the following MIPS loop:

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

2.26.1 Assume that the register $t1 isinitialized to the value 10.What is the value in register $s2 assuming $s2 is initially zero?

Solution:

```
$t1 = 10 | $t1 = 10,   $t1 = 9, $t1 = 8, ...
$s2 = 0  | $s2 =(0+2), $s2 = 4, $s2 = 6, ...
final value of $s2 = 10 * 2 = 20
```

2.26.2 For the loop above, write the equivalent C code routine. Assume that the registers \$s1, \$s2, \$t1, and \$t2 are integers A, B, i, and temp, respectively.

Solution:

```
while ( i > 0 )
  {
    i--;
    B += 2;
  }
```

2.26.3 For the loop written in MIPS assembly above, assume that the register \$t1 is initialized to the value N. How many MIPS instructions are executed?

Solution:

```
//These two instructions are executed when $t2 is initialized as 0
slt $t2, $0, $t1
beq $t2, $0, DONE
```

The remaining 3 instructions are executed for every n > 0

$$(3 + 2)N + 2$$

$$5N + 2$$

2.27 Translate the following C code to MIPS assembly code. Use a minimum number of instructions. Assume that the values of a, b, i, and j are in registers $s0, $s1, $t0, and $t1, respectively. Also, assume that register $s2 holds the base address of the array D.

```
for(i=0; i<a; i++)
      for(j=0; j<b; j++)
          D[4*j] = i + j;
```

Solution:

```
       addi $t0, $0, 0 // i = 0
       addi $t1, $t0, 0 //j = 0
LOOP1: slt $t2, $t0, $s0 //i < a
       beq $t2, $0, EXIT // i !< a br EXIT
       addi $t0, $t0, 1
LOOP2: slt $t2, $t1, $s1 // j < b
       beq $t2, $0, LOOP1 // j !< b go to next iteration of i
       addi $t1, $t1, 1 // j++
       add $t2, $t0, $t1 // $t2 = i + j
       sll $t3, $t1, 2 //$t3 = j*4
       add $t3, $t3, $s2 //base addr of D + 4 * j
       sw $t2, 0($t3) //D[4*j] = $t2 = i + j
       j LOOP2
EXIT:
```

2.31 Implement the following C code in MIPS assembly. What is the total number of MIPS instructions needed to execute the function?

```
unsigned int fib(unsigned int n)
{
  if (n==0)
    return 0;
  else if (n == 1)
    return 1;
  else
    return fib(n−1) + fib(n−2);
}
```

MIPS Assembly of Above Code:

```
FIB:    addi $sp $sp, -16
        sw $a0, 0($sp)
        sw $ra, 4($sp)
        sw $t0, 8($sp)
        sw $t1, 12($sp)
        bne $a0, $0, ELSEIF //n==0, return $v0 = 0
        add $v0 $v0, $0
ELSEIF: addi $t0, $0, 1 //$t0 == 1 for comparison
        bne $a0, $t0, ELSe // if $a0 == 1, add to $v0 and EXIT
        addi $v0, $v0, 1
        j EXIT
ELSE:   addi $a0, $a0, -1 //$a0 = n - 1
        jal FIB
        add $t1, $v0, $0 //save fib(n - 1) in $t1
        addi $a, $a, -1 //$a0 now = n - 2
        jal FIB //$v0 now has fib(n-1)
        add $v0, $v0, $t1 // $v0 = fib(n-1) + fib(n-2)
EXIT:   lw $a0, 0($sp)
        lw $ra, 4($sp)
        lw $t0, 8($sp)
        lw $t1, 12($sp)
        addi $sp, $sp, 16
        jr $ra
```

Number of Insructions Needed to Execute Function:

when n = 0

> 14

when n = 1

> 16

when n = 2

> $10 + fib(1) + 3 + f(0) + 7 = 20 + fib(n-1) + fib(n-2)$

Non-Homogeneous Recurrence Relation:

$f(n) - f(n-1) - f(n-2) = 20$

> replace n with n-1

$f(n-1) - f(n-2) - f(n-3) = 20$

Subtract these two equations: $f(n) - f(n-1) - f(n-2) = 20$

$f(n-1) - f(n-2) - f(n-3) = 20$

Homogeneous Recurrence Relation:

$y(n) - 2y(n-1) + y(n-3) = 0$

this is a homogeneous recurrence relation of order, N = 3

y(0) = 14, y(1) = 16, y(2) = 50

$\lambda^n - 2\lambda^{n-1} + \lambda^{n-3} = 0$

$\lambda^{n-3}(\lambda^3 - 2\lambda^2 + 1) = 0$

> $\lambda^{n-3} = 0$ Trivial Solution

after factoring the remaining equation and solving quadratic formula:

$\lambda_1 = 1$ with multiplicity $m = 1$

$\lambda_2 = \frac{1+\sqrt{5}}{2}$ with $m = 1$

$\lambda_3 = \frac{1-\sqrt{5}}{2}$ with $m = 1$

Closed form for y(n) will be of form:

$y(n) = C_1\lambda_1^n + C_2\lambda_2^n + C_3\lambda_3^n$

Substituting in λ_1 - λ_3

$$y(n) = C_1(1)^n + C_2(\frac{1+\sqrt{5}}{2})^n + C_3(\frac{1-\sqrt{5}}{2})^n$$

System of Equations to Solve:

$$y(0) = C_1 1 + C_2 1 + C_3 1 = 14$$

$$y(1) = C_1(1) + C_2(\frac{1+\sqrt{5}}{2}) + C_3(\frac{1-\sqrt{5}}{2})$$

$$y(2) = C_1(1) + C_2(\frac{1+\sqrt{5}}{2})^2 + C_3(\frac{1-\sqrt{5}}{2})^2$$

$$\begin{pmatrix} 1 & 1 & 1 & 14 \\ 1 & \frac{1+\sqrt{5}}{2} & \frac{1-\sqrt{5}}{2} & 16 \\ 1 & (\frac{1+\sqrt{5}}{2})^2 & (\frac{1-\sqrt{5}}{2})^2 & 50 \end{pmatrix}$$

This is solved using the normal method of Gauss-Jordan Elimination to reduce this to Row-Echelon Form

The work for this is attached at the end as it is quite long

Final Answer:

$$y(n) = (-20)(1)^n + (\frac{36+\frac{104}{sqrt(5)}}{1+\sqrt(5)})(\frac{1+\sqrt(5)}{2})^n$$

3.34 Translate function f into MIPS assembly language. If you need to use registers $t0 through $t7, use the lower-numbered registers first. Assume the function declaration for func is “int func(int a, int b);”. The code for function f is as follows:

```
int f(int a, int b, int c, int d)
{
  return func(func(a,b),c+d);
}
```

Solution:

```
FUNC: ...

F:     addi $sp, $sp, -24 //make space on stack
       sw $t0, 0($sp)
       sw $a0, 4($sp)
       sw $a1, 8($sp)
       sw $a2, 12($sp)
       sw $a3, 16($sp)
       sw $ra, 20($sp)

       add $t0, $a2, $a3 //$t0 = c + d
       jal FUNC //$v0 = func($a0, $a1) = func(a, b)
       add $a0, $v0, $0 //$a0 = func(a, b)
       add $a1, $t0, $0 //$a1 = c + d
       jal FUNC //$v0 = func($a0, $a1) = func( func(a, b), c + d )

       lw $t0, 0($sp)
       lw $a0, 4($sp)
       lw $a1, 8($sp)
       lw $a2, 12($sp)
       lw $a3, 16($sp)
       lw $ra, 20($sp)
       jr $ra
```

2.46 Assume for a given processor the CPI of arithmetic instructions is 1, the CPI of load/store instructions is 10, and the CPI of branch instructions is 3. Assume a program has the following instruction breakdowns: 500 million arithmetic instructions, 300 million load/store instructions, 100 million branch instructions.

2.46.1 Suppose that new, more powerful arithmetic instructions are added to the instruction set. On average, through the use of these more powerful arithmetic instructions, we can reduce the number of arithmetic instructions needed to execute a program by 25%, at the cost of increasing the clock cycle time by only 10%. Is this a good design choice? Why?

Solution:

note: m here represents million, b represents billion

New Number of Arithmetic Operations: $(500m) - (500m * 0.25) = 375m$

Total Execution Time $= IC * CPI_{avg} * T_{clock}$

$IC_{old} = 500m + 300m + 100m = 900m$

$CPIavgOld = (\frac{500}{900} * 1 + \frac{300}{900} * 10 + \frac{100}{900} * 3) \approx 4.19$

$T_{clockOld} = T$

$TotalExecutionTime_{old} = (900m) * (4.19) * T \approx 3.771Tb$

$IC_{new} = 367m + 300m + 100m = 775m$

$CPI_{avgNew} = \frac{375}{775} * 1 + \frac{300}{775} * 10 + \frac{100}{775} * 3 \approx 4.77$

$T_{clockNew} = T + 0.1 = 1.1T$

$TotalExecutionTime_{new} = 775 * 4.77 * 1.1T \approx 4.07Tb$

$Speedup = \frac{oldTime}{newTime} = \frac{3.77}{4.07} = 0.93$

Final Answer:
This would not be a good choice as the new total execution time is slower than the old execution time $newTime \approx 1.075 * oldTime$

2.46.2 Suppose that we find a way to double the performance of arithmetic instructions. What is the overall speedup of our machine? What if we find a way to improve the performance of arithmetic instructions by 10 times?

Solution:

If performance is being doubled/ improved 10x means that the cpi is $\frac{1}{2} * CPI_{old}$ / $\frac{1}{10} * CPI_{old}$

From previous question:

- $TotalExecutionTime_{old} \approx 3.77Tb$

When performance improved by 2x:

$$CPI_{avgNew} = \frac{500}{900} * \frac{1}{2} + \frac{300}{900} * 10 + \frac{100}{900} * 3) \approx 3.91$$

$$TotalExecutionTime_{new} = 900 * 3.91 * T \approx 3.519Tb$$

Speedup $= \frac{3.77}{5.19} \approx 1.07$

When performance improved by 10x:

$$CPI_{avgNew} = \frac{500}{900} * \frac{1}{10} + \frac{300}{900} * 10 + \frac{100}{900} * 3) \approx 3.317Tb$$

Speedup $= \frac{3.77}{3.17} \approx 1.14$

Colab paid products - Cancel contracts here