EE116C/CS151B
Fall 2017

Instructor: Professor Lei He
TA: Yuan Liang

Discussion of Homework

Problem 1

Consider three processors P1, P2 and P3 executing the same instruction set. P1 has a 3GHz clock rate and a CPI of 1.5. P2 has a 2.5GHz clock rate and a CPI of 1.0. P3 has a 4.0GHz clock rate and a CPI of 2.2.

a. Which processor has the highest performance expressed in the instructions per second.

b. If the processors each execute a program in 10 seconds, what is the number of cycles and the number of instructions.

c. We are trying to reduce the time by 30% but this leads to an increase of 20% in the CPI. what clock rate should we have to get this time reduction?

Discussion of Homework

a. Which processor has the highest performance expressed in the instructions per second.

Instruction/second = 1 / cycles/instruction * cycles/second

b. If the processors each execute a program in 10 seconds, int he number of cycles and the number of instructions.

cycles counts = CPU time * cycles/second
Instruction counts = CPU time * cycles/second * instruction/cycles

c. We are trying to reduce the time by 30% but this leads to an increase of 20% in the CPI. what clock rate should we have to get this time reduction?

CPU time = instructions / (instruction/cycles * cycles/second)
CPU time (new)/CPU time (old) = instruction/cycles (old) / instruction/cycle (new) * cycles/second (old)/ cycles/second (new) = 0.7;
thus, clock rate (new) / clock rate (old) = 1.2/0.7 = 1.7143;

Discussion of Homework

Problem 2

Consider two different implementations of the same instruction set architecture. The instructions can be divided into four classes according to their CPI (class A, B, C and D). P1 with a clock rate of 2.5 GHz and CPIs of 1,2,3, and 3. P2 with a clock rate of 3GHz and CPIs of 2,2,2 and 2. Given a program with a dynamic instruction count of 1.0E6 instructions divided into classes as follows: 10% class A, 20% class B, 50% class C and 20% class D.

a. Which implementation is faster?

b. what is the average CPI for each implementation.

c. Find the clock cycles required in both cases.

Discussion of Homework

a. what is the global CPI for each implementation.

P1: global CPI = 1*10% + 2*20% + 3*50% + 3*20% = 2.6
P2: global CPI = 2*10% + 2*20% + 2*50% + 2*20% = 2

b. Find the clock cycles required in both cases.

clock cycles = global CPI * 10^6 instructions
so for:
P1 = 2.6*10^6
P2 = 2*10^6

c. Which implementation is faster?
CPU time = clock cycles * time / cycle = clock cycles / clock rate

Discussion of Homework

problem 3

Compilers can have a profound impact on the performance of an application. Assume that for a program, compiler A results in a dynamic instruction count of 1.0E9 and has an execution time of 1.1s, while compiler B results in a dynamic instruction count of 1.2E9 and an execution time of 1.5s.

a. find the average CPI for each program given that the processor has a clock cycle time of 1ns.

b. assume the compiled programs run on two different processors. If the execution times on the two processors are the same, how much faster is the clock of the processor running compiler A's code versus the clock of the processor running compiler B's code?

c. A new compiler is developed that uses only 6.0E8 instructions and has an average CPI of 1.1. What is the speedup of using this new compiler versus using compiler A or B on the original processor?

Discussion of Homework

a. find the average CPI for each program given that the processor has a clock cycle time of 1ns.

execution time = instruction count * cycles/instruction * cycles/second
so CPI = execution time / instruction count / cycles/second

b. assume the compiled programs run on two different processors. If the execution times on the two processors are the same, how much faster is the clock of the processor running compiler A's code versus the clock of the processor running compiler B's code?

execution time = instruction count * cycles/instruction * cycles/second
thus, execution time (A) / execution time (B) = 1
thus,
instruction count * cycles/instruction * cycles/second (A) = instruction count * cycles/instruction * cycles/second (B)

c. A new compiler is developed that uses only 6.0E8 instructions and has an average CPI of 1.1. What is the speedup of using this new compiler versus using compiler A or B on the original processor?

execution time = instruction count * cycles/instruction * cycles/second

Discussion of Homework

problem 4

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?

As for the Jump (j) instruction:

The previous PC is Ox2000 0000, thus it is

$(0010\ 0000\ 0000\ 0000,\ 0000\ 0000\ 0000\ 0000)_2$ in binary address;

The format of Jump is opcode (6 bit) + address (26 bit)

The first 4 digital is [0010], the farthest place you can jump is to set address as Ox 03FF FFFF, with a multiple of 4, the farthest place is [0010] [1111 1111 1111 1111 1111 1111 1100] = Ox 2FFF FFFC

So, it is not possible.

Discussion of Homework

problem 5

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:
```

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

b. 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?

Discussion of Homework

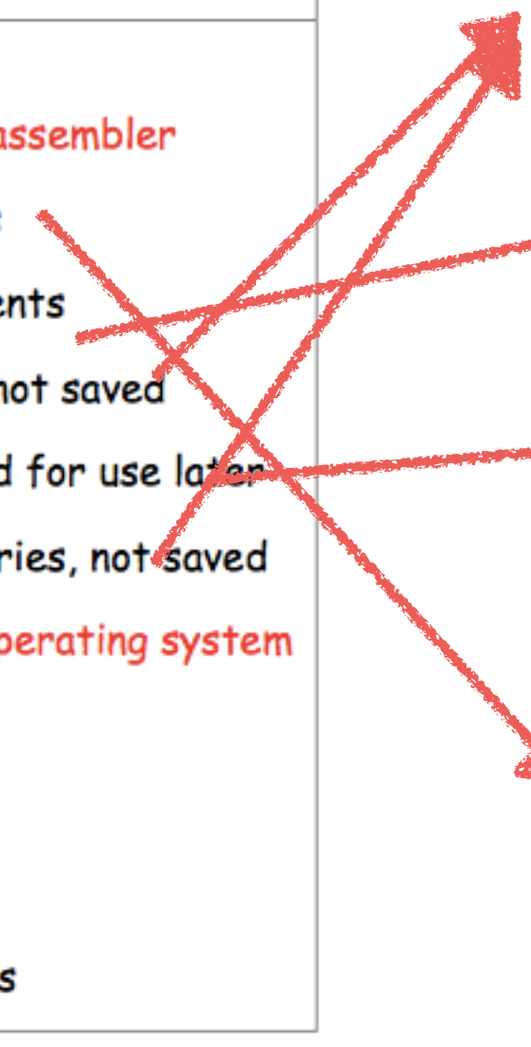
```
do{

if (0<t1) {
   t2=1;
   } else {
   t2 = 0;
   }
   }
if (t2=0)
   break;
t1=t1-1;
s2=s2+2;

} while (1);
```

There are 5 instruction in one loop. As the loop has been executed for N times, and the first two line of the program is executed for 2 more times before the program jump to done, the total instructions executed are: 5N+2.

Review

- **MIPS registers:**

register	assembly name	Comment
r0	$zero	Always 0
r1	$at	Reserved for assembler
r2-r3	$v0-$v1	Stores results
r4-r7	$a0-$a3	Stores arguments
r8-r15	$t0-$t7	Temporaries, not saved
r16-r23	$s0-$s7	Contents saved for use later
r24-r25	$t8-$t9	More temporaries, not saved
r26-r27	$k0-$k1	Reserved by operating system
r28	$gp	Global pointer
r29	$sp	Stack pointer
r30	$fp	Frame pointer
r31	$ra	Return address



– $t0 -- $t9: General purpose (caller temporaries)

– $a0 -- $a3: General purpose (Function arguments)

– $s0 -- $s7: General purpose (callee temporaries)

– $sp: stack pointer

– $v0-$v1 : Return values

– $ra: return address

Review

- **MIPS registers:**

  - $sp : The stack pointer (points to the top of the stack, ie. the lowest address).
  - $fp : The base/frame pointer (points to the base of the current stack frame).

Review

R-instructions:

1. add
   add $t0, $s1, $s2

2. bit and
   and $t0, $t1, $t2

3. bit or
   or $t0, $t1, $t2

4. bit nor
   nor $t0, $t1, $zero

Review

R-instructions:

5. left shift
   sll $t0, $s1, $s2

6. right shift (fill with 0, for unsigned only)
   srl $t0, $s1, $s2

Review

R-instructions:

7. Set result to 1 if a condition is true (conflict?)

slt rd, rs, rt

if (rs < rt) rd = 1; else rd = 0;

8. right shift (fill with 0, for unsigned only)

srl rd, rs, rt

Review

I-instructions:

1. branch if equal
   beq rs, rt, L1
   if (rs == rt) branch to instruction labeled L1;

2. branch if not equal
   bne rs, rt, L1

blt, bge, etc?
No!
Hardware for <, ≥, … slower than =, ≠
Combining with branch involves more work per instruction, requiring a slower clock
All instructions penalized!

Review

I-instructions:

3. save register value to memory according to address
   sw rd, val($rs)

4. read memory to register according to address
   rw rd, val($rs)

Review

I-instructions:

5. add with immediate number
   addi $t0, $s1, 12

6. set result according to immediate number
   slti rt, rs, constant

if (rs < constant) rt = 1; else rt = 0;

Review

J-instructions:

1. Procedure call: jump and link
   jal ProcedureLabel

   Address of following instruction put in $ra
   Jumps to target address

2. Procedure return: jump register
   jr $ra

   Copies $ra to program counter

Review

J-instructions:

3. Unconditional jump to instruction labeled L1

```
j L1
```

Review

MIPS assembly language

Category	Instruction	Example	Meaning	Comments
Arithmetic	add	`add $s1, $s2, $s3`	`$s1 = $s2 + $s3`	Three operands; data in registers
	subtract	`sub $s1, $s2, $s3`	`$s1 = $s2 - $s3`	Three operands; data in registers
	add immediate	`addi $s1, $s2, 100`	`$s1 = $s2 + 100`	Used to add constants
Data transfer	load word	`lw $s1, 100($s2)`	`$s1` = Memory[`$s2` + 100]	Word from memory to register
	store word	`sw $s1, 100($s2)`	Memory[`$s2` + 100] = $s1	Word from register to memory
	load byte	`lb $s1, 100($s2)`	`$s1` = Memory[`$s2` + 100]	Byte from memory to register
	store byte	`sb $s1, 100($s2)`	Memory[`$s2` + 100] = $s1	Byte from register to memory
	load upper immediate	`lui $s1, 100`	$s1 = 100 * 2^{16}	Loads constant in upper 16 bits
Conditional branch	branch on equal	`beq $s1, $s2, 25`	if (`$s1 == $s2`) go to PC + 4 + 100	Equal test; PC-relative branch
	branch on not equal	`bne $s1, $s2, 25`	if (`$s1 != $s2`) go to PC + 4 + 100	Not equal test; PC-relative
	set on less than	`slt $s1, $s2, $s3`	if (`$s2 < $s3`) `$s1` = 1; else `$s1` = 0	Compare less than; for beq, bne
	set less than immediate	`slti $s1, $s2, 100`	if (`$s2 < 100`) `$s1` = 1; else `$s1` = 0	Compare less than constant
Uncondi-tional jump	jump	`j 2500`	go to 10000	Jump to target address
	jump register	`jr $ra`	go to `$ra`	For switch, procedure return
	jump and link	`jal 2500`	`$ra` = PC + 4; go to 10000	For procedure call

Sample question 1

C code:

```
if (i == j) f = g + h;
else f = g - h;
```

i in $s3
j in $s4
Result saves in $s0

Branch

What are the MIPS instructions?

Sample question 1

```
        beq $s3, $s4, True    # Branch if i == j
        sub $s0, $s1, $s2     # f = g - h
        j Exit                # Go to Exit
True: add $s0, $s1, $s2 # f = g + h
Exit:
```

Sample question 2

C code:

```
int sum(int a, int b){
 int result;
 result= a+ b;
 return result;
}
```

a in $a0 (why?)
b in $a1 (why?)
Result saves in $v0 (why?)

calling

What are the MIPS instructions?

Sample question 2

Steps:

1. save current local variables into stack;
2. perform operations;
3. save results to proper registers;
4. restore the caller variables;
5. return to caller.

Sample question 2

```
leaf_example:
   addi $sp, $sp, -4
   sw   $s0, 0($sp)
   add  $t0, $a0, $a1
   add  $t1, $a2, $a3
   sub  $s0, $t0, $t1
   add  $v0, $s0, $zero
   lw   $s0, 0($sp)
   addi $sp, $sp, 4
   jr   $ra
```

Save $s0 on stack

Procedure body

Result

Restore $s0

Return

Sample question 3

C code:

```
int fact (int n)
{
if (n < 1) return f;
else return n * fact(n - 1);
}
```

Argument n in $a0
Result saves in $v0

What are the MIPS instructions?

Reclusion

Sample question 3

Steps:

1. save the current framework to stack (local variable, program pointer).

2. branch on condition

3. set arguments for calling other program, than jump to that program.

4. set results for current program.

5. jump to caller program, by setting the program pointer and restore the program framework for caller.

Sample question 3

MIPS code:

```
fact:
    addi $sp, $sp, -8       # adjust stack for 2 items
    sw   $ra, 4($sp)        # save return address
    sw   $a0, 0($sp)        # save argument
    slti $t0, $a0, 1        # test for n < 1
    beq  $t0, $zero, L1
    addi $v0, $zero, 1      # if so, result is 1
    addi $sp, $sp, 8        #   pop 2 items from stack
    jr   $ra                #   and return
L1: addi $a0, $a0, -1       # else decrement n
    jal  fact               # recursive call
    lw   $a0, 0($sp)        # restore original n
    lw   $ra, 4($sp)        #   and return address
    addi $sp, $sp, 8        # pop 2 items from stack
    mul  $v0, $a0, $v0      # multiply to get result
    jr   $ra                # and return
```