CSCE 5610 – Computer System Architecture

Assignment 1

(100 points)

Question 1 (30 points) Instruction set architecture. For the following, we consider the instruction encoding for instruction set architecture.

a. Consider the case of a processor with an instruction length of 14 bits and with 64 general-purpose registers so the size of the address fields is 6 bits. Is it possible to have instruction encodings for the following?
   3 two-address instructions
   63 one-address instructions
   45 zero-address instructions

b. Assuming the same instruction length and address field sizes as above, determine if it is possible to have:
   3 two-address instructions
   65 one-address instructions
   35 zero-address instructions

c. Assume the same instruction length and address field sizes as above. Further assume there are already 3 two-address and 24 zero-address instructions. What is the maximum number of one-address instructions that can be encoded for this processor?

Answer:

The length of the instruction is 14 bits. There are 64 general-purpose registers, thus the size of the address fields is 6 bits for each register operand. Since there are instructions with two-addresses, then the 2 most significant bits are reserved for opcodes. The notation [13:0] is used to represent the 14 bits of an instruction.

a. Possible.(10pts) First, we need to support 3 two-addresses instructions. These can be encoded as follows:

	addr[13:12]	addr[11:6]	addr[5:0]
3 two-address instructions	"00", "01", "10"	"000000"to"111111"	"000000"to"111111"
Other encodings	"11"	"000000"to"111111"	"000000"to"111111"

Hence, the one-address and zero-address instructions must be encoded using addr [13:12]="11". For the one-address instructions, the opcode is extended by using 63 of the possible 64 combinations from the bits in addr[11:6], that is, from "000000" to "111110". Consequently, the opcode of zero-address instructions is extended with the remaining combination of "111111"

from addr[11:6]. There are 45 zero-address instructions, so, using “000000” to “101100” from addr[5:0] suffices for the encoding.

	addr[13:12]	addr[11:6]	addr[5:0]
3 two-address instructions	“00”, “01”, “10”	“000000”to”111111”	“000000”to”111111”
63 one-address instructions	“11”	“000000”to”111110”	“000000”to”111111”
45 zero-address instructions	“11”	“111111”	“000000”to”101100”
19 unused encodings	“11”	“111111”	“101101to”111111”

b. Impossible. (10pts)

3 two-address instructions can be encoded similar to part(a). The one-address and zero-address instructions must be encoded using addr[13:12] = “11”. In order to encode 65 one-address instructions, at least 7 bits are required, as add[11:5] can be used for this encoding. The addr[4:0] is left for register address. But register address field need 6 bits. So, impossible.

c. Similar to part (a), the 3 two-address instructions can be encoded as follows:

	addr[13:12]	addr[11:6]	addr[5:0]
3 two-address instructions	“00”, “01”, “10”	“000000”to”111111”	“000000”to”111111”

The one-address and zero-address instructions must be encoded using addr[13:12]=”11”. One-address instructions can be encoded using all but one of the possible combinations from addr[11:6], that is, from “000000” to “111110”. The “111111” combination is used to extend the opcode for the zero-address instructions. The maximum number of one-address instructions that can be encoded for this processor is 2^6-1=63(10pts).

Question 2 (40 points) Convert each of the below C code snippet to MIPS assembly code. Comment your assembly code.

1). (5 points) Assume variable a and b is stored in registers $t0 and $t1, and are 32-bits non-zero positive integer. Base address of c is stored in register $s0. Comment your assembly code:

$a = a - b$

$c[0] = b + 1$

$c[a + 2] = c[0] - c[b + 1]$

2). (25 points) Assume variables a, b, and c are stored in registers $t0, $t1, and $t2 respectively and are 32-bits non-zero positive integer. Base address of d is stored in register $s0. Do not use multiply and divide instruction (hint: use shift left logical (sll) for multiplication and shift right logical (srl) for division).

a). (8 points) if$(b \leq c)$ $d[b] = a/4$

else $d[b] = a + b * 2$

b). (8 points) for(i=0;i<a;i++)

```
    d[i]=d[b]-2*i
```

c). (9 points) i=0;

```
while(d[i]>0){
    d[i] = d[a] + 4*i;
    i++;
}
```

3). Assume variables n is stored in register $a0, and is 32-bits non-zero positive integer. x is stored in $t0 and the return value from function fib should be saved in $v0.

```
int fib(int n)
{
    if (n <= 1){
        return n;
    }
    else{
        return fib(n-1)*fib(n-2);
    }
}
int main()
{
    int n = 3;
    int x = fib(n);
    return 0;
```

}

Answer:

1). (0.5pts/instruction)

```
    sub $t0, $t0, $t1 //a=a-b
    addi $t1, $t1, 1 //t1=b+1
    sw $t1, 0($s0) //Mem[$s0+0] = $t1
    sll $t2, $t1, 2 //t2=(b+1)*4
    add $t2, $t2, $s0 //c[b+1] addr=base addr + (b+1)*4
    lw $t2, 0($t2) //t2 = c[b+1]
    sub $t2, $t1, $t2 //t2=c[0]-c[b+1]
    addi $t0, $t0, 2 //a=a+2
    sll $t0, $t0, 2 //t0=(a+2)*4
    add $t0, $t0, $s0 //c[a+2] addr=base addr + (a+2)*4
    sw $t2, 0($t0) //Mem[$t0+0] = $t2
```

2).

a). (1 point/instruction)

```
    sll $t3, $t1, 2 //t3 = b*4
    add $t3, $t3, $s0 // d[b] addr = base addr + b*4
    bgt $t1, $t2, E //if b> c, go to E
    srl $t4, $t0, 2 //t4 = a/4
    sw $t4, 0($t3) //Mem[$t3+0] = t4
    j   Exit
E: sll $t1, $t1, 1 // t1 = b*2
   add $t4, $t0, $t1 // t4 = a+b*2
   sw $t4, 0($t3) // Mem[$t3+0] = t4
Exit:
```

b). (1 point/instruction)

```
    add  $t9, $zero, $zero //i is initialized to 0, $t9 = 0
    sll $t1, $t1, 2 //t1=b*4, offset of d[b]
    add $t1, $t1, $s0 // d[b] addr = base addr + b*4
    lw $t1, 0($t1) // load d[b] to t1
L:  sll $t8, $t9, 1 //t8 = 2*i
    sub $t8, $t1, $t8 //t8 = d[b]-2*i
    sll $t2, $t9, 2 // t2=i*4, offset of d[i]
    add $t2, $t2, $s0 // d[i] addr = base addr + i*4
```

```
    sw $t8, 0($t2) //store d[b]-2*i to d[i]
    addi $t9, $t9, 1 // i++
    slt $t7, $t9, $t0 // t7 = 1 if i < a
    bne $t7, $0, L //go to loop if i<a
```

c). (1 point/instruction)

```
    add  $t9, $zero, $zero //i is initialized to 0, $t9 = 0
    sll $t0, $t0, 2 //t1=a*4, offset of d[a]
    add $t0, $t0, $s0 // d[a] addr = base addr + a*4
    lw $t0, 0($t0) // load d[a] to t0
W: sll $t7, $t9, 2 // t7=i*4, offset of d[i]
    add $t2, $t7, $s0 // d[i] addr = base addr + i*4
    lw $t3, 0($t2) // load d[i] to t3
    ble $t3, $0, Exit //if d[i] <= 0, exit
    add $t8, $t0, $t7 // t8 = d[a] + i*4
    sw $t8, 0($t2) // store t8 to d[i]
    j       W
Exit:
```

3). (0.5pts/instruction)

```
li $a0, 3; //n=3
subi $sp, $sp, 4;
sw $a0, 0($sp);
jal Fib;
lw $a0, 0($sp);
addi $sp, $sp, 4;
move $s0, $v0; //x=fib(3)

Fib:
        subi $sp, $sp, 8;
        sw $ra, 4($sp);
        sw $a0, 0($sp)
        blei $a0, 1, E1 //if(n<=1), return n

        //fib(n-1)
        subi $a0, $a0, 1
        jal Fib
        move $t1, $v0 //t1=fib(n-1)
```

```
        //fib(n-2)
        lw $a0, 0($sp)
        subi $a0, $a0, 2
        jal fib
        move $t2, $v0 //t2= fib(n-2)

        //fib(n-1)*fib(n-2)
        mul $v0, $t1, $t2
        j E2;
E1:
    move $v0, $a0; //return n
E2:
    lw $ra, 4($sp);
    lw $a0, 0($sp);
    addi $sp, $sp, 8
    jr $ra;
```

Question 3 (30 points). Assume that there is an integer array c with n integers in it. Its base address is saved in register $s0 and the number of elements n is saved in register $t1. Decode the following machine language (convert the machine language to assembly language). Let the first instruction be at address $0x00004000_{hex.}$ And briefly explain what this code do.

Here are eight machine language instructions in hexadecimal:

0X20080000

0X11090006

0X00085042

0X00085880

0X01705820

0XAD6A0000

0X21080001

0X08001001

Answer:

1. Convert the hex to binary:

0010,00 00,000 0,1000, 0000,0000,0000,0000

0001,00 01,000 0,1001, 0000,0000,0000,0110

0000,00 00,000 0,1000, 0101,0000,0100,0010

0000,00 00,000 0,1000, 0101,1000,1000,0000

0000,00 01,011 1,0000, 0101,1000,0010,0000

1010,11 01,011 0,1010, 0000,0000,0000,0000

0010,00 01,000 0,1000, 0000,0000,0000,0001

0000,10 00,000 0,0000, 0001,0000,0000,0001 (5 points, 0.5/each)

2. Identify the opcode and format

R	0	rs	rt	rd	shamt	funct
I	1, 4-62	rs	rt	immediate		
J	2 or 3	target address				

Look at opcode: 0 means R-Format, 2 or 3 mean J-Format, otherwise I-Format. So, we have I (8), I (4), R(0), R(0), R(0), I(43), I(8), J(2).

3. Fields separated based on format/opcode:

8	0	8	0
4	8	9	6

0	0	8	10	1	2
0	0	8	11	2	0
0	11	16	11	0	32
43	11	10	0		
8	8	8	1		

2	4097

4. translate to assembly code:

0x00004000 addi $t0, $0, 0

0x00004004 **Loop:** beq $t0, $t1, Exit

0x00004008 srl $t2,$t0, 1

0x0000400c sll $t3,$t0, 2

0x00004010 add $t3, $t3, $s0

0x00004014 sw $t2, 0($t3)

0x00004018 addi $t0, $t0, 1

0x0000401c j 0X1001

Exit: (10 points, 1 point/each)

```
t0 = 0;
while(t0!=t1){
    t2=t0/2;
    c[t0]=t2;
    t0++;
}
```

(10 points, 2 points/each)

This code updates all the values in array c to (the corresponding index/2). (5 point)