CSE-312 COMPUTER ARCHITECTURE

Mid-Semester Exam

Name: YASH VERMA CSE B.tech 19075083

1.) The Computer processor can only execute extremly simple low-level instructions. The easiest signals for computers to understand are on and off, symbolically 0 and 1. So, the computer language can be ~~bought~~ thought of as set of binary numbers. So, the high-level like C goes through few steps till it could be converted to binary representation and be directly executed by a computer processor.

The steps that transform C into binary language are:

→ Compilation: It is the translation of a program written in high-level language like C, into assembly language statements, which is a symbolic representation of machine ~~representation~~ instructions.

→ Assemble: It is the translation of assembly language obtained in previous step to binary language or machine language. The program ~~w~~used for executing it is called as assembler.

Example:

High-level Language program

```
swap (int a[], int v)
{   int t;
    t = a[v];
    a[v] = a[v+1];
    a[v+2] = t;
}
```

↓

Compiles

```
swap  multi  $2, $5, 4
      add    $2, $4, $2
      lw     $15, 0($2)
      lw     $17, 4($2)
      sw     $17, 0($2)
      sw     $15, 4($2)
      jr     $32
```

↓

Assembles

Binary (Machine) Language program

```
000.0000101 ..
0000 00000 .
10 00111001111 ...
1000 100010101 ...
.
.
.
00000011111 ...
```

Scanned with CamScanner

- other than this 2 other software was also used in previous time.

(i) Linker: Link all code together & create an executable file.

(ii) Loader: Load all the library functions into the code.

- In Nowdays, we don't have to use them manually. They are now a part of the compiler & assembler itself.

2. Execution time$_A$ = 10 s

Execution time$_B$ = 15 s

$$\text{Performance}_A = \frac{1}{\text{Execution time}_A}$$

$$= \frac{1}{10}$$

$$\text{Performance}_B = \frac{1}{\text{Execution time}_B} = \frac{1}{15}$$

Clearly, Performance$_A$ > Performance$_B$ i.e. $\frac{1}{10} > \frac{1}{15}$.

So, A is faster than B.

$$\frac{\text{Performance}_A}{\text{Performance}_B} = \frac{1}{10} \times \frac{15}{1} = \frac{15}{10}$$

$$\text{Performance}_A = \frac{3}{2} \cdot \text{Performance}_B$$

$$\boxed{\text{Performance}_A = (1.5) \times \text{Performance}_B}$$

A is 1.5 times faster than B

3. MIPS Assembly Code:

```
        addi    $t0, $0, 0
        beq     $0 , $0, TEST 1

LOOP1:  addi    $t1, $0, 0
        beq     $0 , $0, TEST2

LOOP2:  add     $t3, $t0, $t1
        sll     $t2, $t1, 4
        add     $t2, $t2, $s2
        sw      $t3, (:$t2)
        addi    $t1 , $t1, 1

TEST2:  slt     $t2, $t1, 1
        bne     $t2, $0, LOOP2
        addi    $t0 , $t0, 1

TEST1:  slt     $t2, $t0, $s0
        bne     $t2, $0, LOOP1
```

Explaination of code:

1.) Let $t0 as zero i.e. i=0

2.) Jump to loop condition (can also use j LoopA)

3.) Set $t1 as zero i.e. j=0.

4.) Jump to loop condition (can also use j LoopB)

5.) $t3 = $t0 + $t1 , (i+j) is stored in $t3

6.) Left shift $t1 by 4 and ~~save~~ save in $t2.

7.) $t2 = $t2 + $s2 i.e. $t2 = &D[4*j]

8.) ~~Compare~~ Save $t3(i+j) in $t2.

9.) Increment $t1 (j++)

10.) Compare $t1 with $s1 (i.e if j<1 $t1 = 1
otherwise $t2 = 0)

11.) If $t2 != 0, jump to Loop2

12.) Increment $t0 (i++)

13.) Compare $t0 with $s0 (i.e if it a $t2 = 1
otherwise $t2 = 0)

14.) If $t2 != 0, jump to Loop1.

4. $s0 = f , $s1 = g , $s2 = h , $s3 = i , $s4 = j

$s6 holds base address of A and $s7 holds base address of B.

Assembly:

```
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      # $t2 = A[f+1]
lw    $t0 , 0($t2)       # $t0 = A[f+1]
add   $t0 , $t0 , $s0    # $t0 = A[f+1] + A[f]
sw    $t0 , 0($t1)       # B[g] = A[f+1] + A[f]
```

Corresponding C language:

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

The two statements cannot swap order

5. ⇒ i → $t1

result → $s2

$s0 → base address of array Memarray

```
addi    $t1, $0, $0
Loop:   lw   $s1, 0($s0)
add     add  $s2, $s2, $s1
        addi $s0, $s0, 4
        addi $t1, $t1, 1
        slti $t2, $t1, 100
        bne  $t2, $s0, Loop
```

Going over code line by line:

1.) Set $t1 = 0

There is a syntax error it can be written of these 2:

add $t1, $0, $0 or addi $t1, $0, 0

2.) Load first element of array in $s1

3) $s2 = $s2 + $1

i.e. result = result + Memarray[i]

4.) Increment $s0 by 4, now it will point to the next element in array.

5.) Increment $t1 i.e. (i++)

6.) If $t1 < 100 : t2 = 1

else t2 = 0

7.) There may be an error. In case of bne $t2, $s0, Loop it means.

if $t2 != $s0, jump to Loop.

It will never happen as $t2 is initially 0 and incremented by 1, while $s0 is incremented by 4. So, this will result in infinite loop.

Instead Consider:

bne $t2, $s0 Loop

If $t2 != 0, jump to Loop

So, final code will be :-

```
for (i=0; i<100; i++){
    result = result + Memarray[i];
}
```

6. Given:

Value of $t0 → 0x20000000.

Instructions:

```
slt   $t2, $0, $t0
beq   $t2, $0, ELSE
 j    done
ELSE: addi  $t2, $t2, 2
DONE:
```

→ for: slt $t2, $0, $t0

if ($0 < $t0) then $t2 = 1 else $t2 = 0

So, ∴ $t2 = 1 (as value of $t0 : 0x20000000)

→ for: beq $t2, $0, ELSE

if ($t2 = = $0) go to ELSE statement skipping

All statements in between

But $t2 > $0 (as $t2 = 1)

So, it will not go to ELSE

Value of $t2 = 1

→ for: j DONE

jump to DONE

It will jump directly to DONE ignoring the instruction

add i $t2, $t2, 2

∴ final value of $t2 = 1