`

UIU QUESTION BANK

MID-TERM QUESTION SOLUTIONS

COMPUTER ARCHITECTURE

CSE 3313

SOLUTION BY

NURUL ALAM ADOR

UPDATED TILL ***SPRING 2024***

Index

Spring 2024

1. Consider the following MIPS code:

```
block:
    add $sp, $sp,-8
    sw $ra, $zero($sp)
    sw $s0, 4($sp)

    addi $v0, 0, 0
    addi $s0, $zero, 1

loop:
    slti $t0, $a0, s0
    bne $t0, $zero, end_loop
    add $v0, $v0, $s0
    addi $s0, $s0, 1
    jump loop

end_loop:
    lw $ra, 4($sp)
    lw $s0, 8($sp)
    addi $sp, $sp, 12
    jr $ra
```

Instruction	CPI
add	2
addi	3
sub	2
lw	4
sw	4
sll	1
srl	1
slt	4
slti	8
beq	4
bne	4
j	8
jr	4
jal	4

Table 1: CPI for each instruction for program 1a

[N.B. **You should assume the value of $a0 = 3**]

1. a) **Find** the errors and fix them in the above code.

Solution:

The code has been rewritten below with error correction:

```
block:
    addi $sp, $sp,-8          # addi should use for constant operand
    sw   $ra, 0($sp)          # offset of sw should be constant
    sw   $s0, 4($sp)

    addi $v0, $zero, 0        # only last operand is constant in addi
    addi $s0, $zero, 1

loop:
    slt  $t0, $a0, $s0        # last operands of slti cannot be register
    bne  $t0, $zero, end_loop
    add  $v0, $v0, $s0
    addi $s0, $s0, 1
    j    loop                 # jump is not a valid instruction

end_loop:
    lw   $ra, 0($sp)          # lw  addresses doesn't match with sw
    lw   $s0, 4($sp)
    addi $sp, $sp, 8          # 8 was allocated for $sp
    jr   $ra
```

1. b) As you get error-free code from 1a, **Write** the value of \$vo and \$s0 registers after executing the program.

Solution:

The value of the following registers has been written below:

```
$v0 = 6
$s0 = 4
```

1. c) Consider a CPU with a *4GHz* clock rate and CPI in the following table-1. **Calculate** the CPU time for executing the program 1a.

Solution:

Given,

$$\text{CPU Clock Rate} = 4\ GHz = 4 \times 10^9\ Hz$$

Here,

Instruction	addi	sw	slt	bne	add	j	lw	jr
Instuction Count (IC)	7	2	4	4	3	3	2	1
CPI	3	4	4	4	2	8	4	4

Now,

$$\text{Clock Cycle} = \text{IC} \times \text{CPI}$$
$$= 7 \times 3 + 2 \times 4 + 4 \times 4 + 4 \times 4 + 3 \times 2 + 3 \times 8 + 2 \times 4 + 1 \times 4$$
$$= 103$$

We know,

$$\text{CPU Time} = \text{Clock Cycle} \times \text{Clock Time}$$
$$= \frac{\text{Clock Cycle}}{\text{Clock Rate}}$$
$$= \frac{103}{4 \times 10^9}$$
$$= 2.575 \times 10^{-8}\ s$$

1. d) Consider program 1a running on another computer that requires 160ns, with 40ns spent executing FP instructions, 90ns executed L/S instructions, and 30ns spent executing branch instructions. What is the improvement factor using Amdahl's law if we only improve the performance of L/S instructions using a better ALU to get the program completion time improved by 2x?

Solution:

Here,

$T_{affected} = 90\ ns$ [L/S instructions time]

$$T_{uneffected} = (160 - 90)\ ns$$
$$= 70\ s$$

$$T_{improved} = \frac{160}{2}\ ns$$
$$= 80\ ns$$

Improvement factor, $n = ?$

We know,

$$T_{improved} = \frac{T_{affected}}{n} + T_{unaffected}$$

or, $80 = \frac{90}{n} + 70$

or, $\frac{90}{n} = 10$

or, $\frac{90}{10} = n$

$\therefore n = 9$

2. Consider the following C function. Compiler will assign a (base address) into \$s0, b (base address) into \$s1, x into \$s2 and i into \$s3.

```c
int main() {
    int x=1,i=32,a[10],b[10];
    do{
        if(a[2*i]==b[2*(i+1)]){
            x += a[2*i]- b[2*(i+1)];
            break;
        }
        else if(a[2*i] & b[2*(i+1)]){
            i = i+x;
            continue;
        }
        else{
            i = i- 3;
        }
        i = i+1;
    }while(i%4);
    return 0;
}
```

[N.B. **You should assume the value of \$a0 = 3**]

2. a) **Convert** the code to the corresponding MIPS assembly instructions.

Solution:

The code has been converted into MIPS assembly instructions:

```
main:
    addi $sp, $sp, -16
    sw   $s0, 12($s0)
    sw   $s1, 8($s0)
    sw   $s2, 4($s0)
    sw   $s3, 0($s0)
    addi $s2, $zero, 1
    addi $s3, $zero, 32

loop:
    sll  $t0, $s3, 2
    add  $t0, $s0, $t0
    lw   $t1, 0($t0)
    addi $t0, $s3, 1
    sll  $t0, $t0, 2
    add  $t0, $s1, $t0
    lw   $t2, 0($t0)

if:
    bne  $t1, $t2, else_if
    sub  $t0, $t1, $t2
    add  $s2, $s2, $t0
    j    end

else_if:
    and  $t0, $t1, $t2
    beq  $t0, $zero, else
    add  $s3, $s3, $s2
    j    loop

else:
    addi $s3, $s3, -3
    addi $s3, $s3, 1
    and  $t0, $s3, 3
    bne  $t0, $zero, loop

end:
    sw   $s3, 0($s0)
    sw   $s2, 4($s0)
    sw   $s1, 8($s0)
    sw   $s0, 12($s0)
    addi $sp, $sp, 16
    jr $ra
```

2. b) Convert the first 8 lines of MIPS assembly instructions to the corresponding machine code after entering the loop body. No need to convert it to binary.

Solution:

The code has been converted into MIPS assembly instructions:

1.

op	rs	rt	rd	shamt	funct
0	X	19	8	2	0

2.

op	rs	rt	rd	shamt	funct
0	16	8	8	X	32

3.

op	rs	rt	C/A
35	8	9	0

4.

op	rs	rt	C/A
8	19	8	1

5.

op	rs	rt	rd	shamt	funct
0	X	8	8	2	0

6.

op	rs	rt	rd	shamt	funct
0	17	8	8	X	32

7.

op	rs	rt	C/A
35	8	10	0

8.

op	rs	rt	C/A
5	10	9	else_if/4

9.

op	rs	rt	rd	shamt	funct
0	9	10	8	X	34

10.

op	rs	rt	rd	shamt	funct
0	18	8	18	X	32

2. c) Assume that the processor has 64 registers. The size of MIPS instruction is 32 bits and 6 bits are reserved for opcode. The structure for addi instruction is given in the table-2. **Find** out the maximum constant value for **addi** instruction in MIPS. Note that, MIPS supports negative constants.

opcode	rs	rt	constant

Table 2: Structure of addi instruction

Solution:

We know,
Structure of addi instruction,

opcode	rs	rt	constant
6 bits	5 bits	5 bits	16 bits

We have 16 bits space for constants. Since MIPS supports negative constant, 1 bit used as sign bit and remaining 15 bits store constant value.

$\therefore$ Maximum constant value $= 2^{15} - 1$
$= 32767$

3. Using the division algorithm, **show** each step of the division of 15 by 4.

Solution:

$15 \div 4$

1111 (DN)
0100 (DR)

Initially: A, Q: $0000\ 1111$
M: 0100
-M: 1100

M = 0100
1011
$\underline{+1}$
-M = 1100

Step 1:	A, Q: 0001 111_	A = A-M
	A, Q: 1101 111$\underline{0}$	= 0001+1100
	A, Q: 0001 1110	= 1101
Step 2:	A, Q: 0011 110_	A = A-M
	A, Q: 1111 110$\underline{0}$	= 0011+1100
	A, Q: 0011 1100	= 1111
Step 3:	A, Q: 0111 100_	A = A-M
	A, Q: 0011 100$\underline{1}$	= 0111+1100
		= 0011
Step 4:	A, Q: 0111 001_	A = A-M
	A, Q: 0011 001$\underline{1}$	= 0111+1100
		= 0011

∴ Reminder, A = 0011 (3)
∴ Quotient, Q = 0011 (3)

Fall 2023

1. a) Consider three different processors P1, P2, and P3 executing the same instruction set architecture (ISA). P1 has a 3*GHz* clock rate and a CPI of 1.5. P2 has a 2.5*GHz* clock rate and a CPI of 1.0. P3 has a 4.0*GHz* clock rate and has a CPI of 2.2.

i) **Find** out which processor performs better by calculating CPU time.

ii) If each processor's execute a program in 10 seconds, **find** the number of clock cycles and the number of instructions.

Solution:

i) For processor P1,

$$\text{CPU Time}_{P1} = \frac{\text{IC}_{P1} \times \text{CPI}_{P1}}{\text{Clock Rate}_{P1}} = \frac{I \times 1.5}{3 \times 10^9} = I \times 5 \times 10^{10}\ s$$

For processor P2,

$$\text{CPU Time}_{P2} = \frac{\text{IC}_{P2} \times \text{CPI}_{P2}}{\text{Clock Rate}_{P2}} = \frac{I \times 1}{2.5 \times 10^9} = I \times 4 \times 10^{-10}\ s$$

For processor P3,

$$\text{CPU Time}_{P3} = \frac{\text{IC}_{P3} \times \text{CPI}_{P3}}{\text{Clock Rate}_{P3}} = \frac{I \times 2.2}{4 \times 10^9} = I \times 5.5 \times 10^{10}\ s$$

Here,

$\text{CPI}_{P1} = 1.5$
$\text{Clock Rate}_{P1} = 3\ GHz = 3 \times 10^9\ Hz$
$\text{CPI}_{P2} = 1$
$\text{Clock Rate}_{P2} = 2.5\ GHz = 2.5 \times 10^9\ Hz$
$\text{CPI}_{P3} = 2.2$
$\text{Clock Rate}_{P3} = 4\ GHz = 4 \times 10^9\ Hz$
$\text{IC}_{P1} = \text{IC}_{P2} = \text{IC}_{P3} = I$

Since $\text{CPU Time}_{P1} > \text{CPU Time}_{P2} > \text{CPU Time}_{P3}$,
∴ Processor P2 perform better.

ii) For processor P1,

$$\text{Clock Cycle}_{P1} = \text{CPU Time} \times \text{Clock Rate}_{P1} = 10 \times 3 \times 10^9 = 30 \times 10^9$$

$$\text{IC}_{P1} = \frac{\text{Clock Cycle}_{P1}}{\text{CPI}_{P1}} = \frac{30 \times 10^9}{1.5} = 2 \times 10^{10}$$

Here,

$\text{CPU Time} = 10\ s$

For processor P2,

$$\text{Clock Cycle}_{P2} = \text{CPU Time} \times \text{Clock Rate}_{P2} = 10 \times 2.5 \times 10^9 = 25 \times 10^9$$

$$\text{IC}_{P2} = \frac{\text{Clock Cycle}_{P2}}{\text{CPI}_{P2}} = \frac{25 \times 10^9}{1} = 2.5 \times 10^{10}$$

For processor P3,

$$\text{Clock Cycle}_{P3} = \text{CPU Time} \times \text{Clock Rate}_{P3}$$
$$= 10 \times 4 \times 10^9$$
$$= 40 \times 10^9$$

$$\text{IC}_{P3} = \frac{\text{Clock Cycle}_{P3}}{\text{CPI}_{P3}} = \frac{40 \times 10^9}{2.2} = 1.8 \times 10^{10}$$

1. b) Consider a program running on a computer that requires 280ns, with 80ns spent executing FP instructions, 180ns executed L/S instructions, and 20ns spent executing branch instructions..

i) By how much is the total time reduced if the time for FP operations is reduced by 20?

ii) What is the improvement factor using Amdahl's law if we only improve the performance of L/S instructions using a better ALU to get the program completion time improved by 2x?

Solution:

i) Here,

Previous total time $= 280\ ns$

L/S instruction execution time $= 180\ ns$

Branch instruction execution time $= 20\ ns$

After reducing by $20\ ns$,

New FP instruction execution time $= 80 - 20\ ns$
$= 60\ ns$

$\therefore$ New total time $= 180 + 20 + 60\ ns$
$= 260\ ns$

$\therefore$ Total time reduced $= 280 - 260\ ns$
$= 20\ ns$

ii) Here,

$T_{affected} = 180\ ns$ [L/S instructions time]

$T_{uneffected} = (280 - 180)\ ns$
$= 100\ ns$

$T_{improved} = \dfrac{280}{2}\ ns$
$= 140\ s$

Improvement factor, $n = ?$

We know,

$$T_{improved} = \frac{T_{affected}}{n} + T_{unaffected}$$

or, $140 = \dfrac{180}{n} + 100$

or, $\dfrac{180}{n} = 40$

or, $\frac{180}{40} = n$

$\therefore n = 4.5$

2. Consider the following C function. Assume necessary registers.

```
     2000: int check_func(int a[],int x){
               if(x%2){
                   a[x] = a[x]-x;
                   return a[x];
               }
           }
     3000: int get_sum(int a[],int b[],int k){
               int sum = 0;
               for(int i=0;i<k;i++){
                   if(check_func(a[i+1],i)){
                       b[i] = sum + (a[i]-b[i])/2;
                       sum = b[i];
                   }
               }
               return sum;
           }
           int main() {
 PC->5000:     int res=0,n=10;
               int a[n],b[n];
               res = get_sum(a,b,n);
               res = res*9;
               return 0;
           }
```

2. a) Convert the code to the corresponding MIPS assembly instructions.

Solution:

The code has been converted into MIPS assembly instructions:

```
   5000:  addi $s0, $zero, 0
   5004:  addi $s1, $zero, 10
   5008:  add  $a0, $s2, $zero
   5012:  add  $a1, $s3, $zero
   5016:  add  $a2, $s1, $zero
   5020:  jal  ~~get_sum~~ 3000
   5024:  add  $s0, $v0, $zero
   5028:  sll  $t0, $s0, 3
   5032:  add  $s0, $t0, $s0
   5036:  addi $v0, $zero, 0
   .
   .
get_sum:
   3000:  addi $sp, $sp, -8
   3004:  sw   $s0, 4($sp)
   3008:  sw   $s1, 0($sp)
   3012:  addi $s0, $zero, 0
   3016:  addi $s1, $zero, 0
loop:
```

Assuming,

res in $s0
n in $s1
a in $s2
b in $s3

Assuming,

sum in $s0
i in $s1

```
    3020:  slt  $t0, $s1, $a2
    3024:  beq  $t0, $zero, ~~exit_loop~~ 3144
    3028:  addi $t0, $s1, 1
    3032:  sll  $t0, $t0, 2
    3036:  add  $t0, $a0, $t0
    3040:  lw   $t1, 0($t0)
    3044:  addi $sp, $sp, -12
    3048:  sw   $a0, 8($sp)
    3052:  sw   $a1, 4($sp)
    3056:  sw   $ra, 0($sp)
    3060:  add  $a0, $t1, $zero
    3064:  add  $a1, $s1, $zero
    3068:  jal  ~~check_func~~ 2000
    3072:  lw   $ra, 0($sp)
    3076:  lw   $a1, 4($sp)
    3080:  lw   $a0, 8($sp)
    3084:  addi $sp, $sp, 12
    3088:  beq  $v0, $zero, ~~inc_i~~ 3132
    3092:  sll  $t0, $s1, 2
    3096:  add  $t1, $a0, $t0
    3100:  add  $t2, $a1, $t0
    3104:  lw   $t3, 0($t1)
    3108:  lw   $t4, 0($t2)
    3112:  sub  $t0, $t3, $t4
    3116:  srl  $t0, $t0, 1
    3120:  add  $t0, $s0, $t0
    3124:  sw   $t0, 0($t2)
    3128:  lw   $t0, 0($t2)
    3132:  add  $s0, $t0, $zero
  inc_i:
    3136:  addi $s1, $s1, 1
    3140:  j    ~~loop~~ 3020
  exit_loop:
    3144:  add $v0, $s0, $zero
    3148:  sw   $s1, 0($sp)
    3152:  sw   $s0, 4($sp)
    3156:  addi $sp, $sp, 8
    3160:  jr   $ra
    .
    .
  check_func:
    2000:  andi $t0, $a1, 1
    2004:  beq  $t0, $zero, ~~exit_check~~ 2036
    2008:  sll  $t0, $a1, 2
    2012:  add  $t1, $a0, $t0
    2016:  lw   $t2, 0($t1)
    2020:  sub  $t0, $t2, $a1
    2024:  sw   $t0, 0($t1)
    2028:  lw   $t0, 0($t1)
    2032:  add  $v0, $t0, $zero
  exit_check:
    2036:  jr   $ra
```

2. b) **Convert** the first 12 lines of get_sum() function's assembly instructions to the corresponding machine code. No need to convert it to binary.

Solution:

The code has been converted into MIPS assembly instructions:

1.

op	rs	rt	C/A
8	29	29	-8

2.

op	rs	rt	C/A
43	29	16	4

3.

op	rs	rt	C/A
43	29	17	0

4.

op	rs	rt	C/A
8	0	16	0

5.

op	rs	rt	C/A
8	0	17	0

6.

op	rs	rt	rd	shamt	funct
0	17	6	8	X	42

7.

op	rs	rt	C/A
4	0	8	786

8.

op	rs	rt	C/A
8	17	8	1

9.

op	rs	rt	rd	shamt	funct
0	X	8	8	2	0

10.

op	rs	rt	rd	shamt	funct
0	4	8	8	X	32

11.

op	rs	rt	C/A
35	8	9	0

12.

op	rs	rt	C/A
8	29	29	-12

2. c) Monica claims that the **sll $t0, $s1, 40** instruction is correct in the MIPS architecture, but Joey disagrees with Monica's claim. **Justify** your opinion with a proper explanation.

Solution:

We know,
Structure of sll instruction,

opcode	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

Here,

Maximum value for shamt $= 2^5 - 1 = 31$

Given instruction:

```
  sll $t0, $s1, 40
```

Since shamt (shift amount) of given instruction is greater than maximum value of shamt ($40 > 31$), therefore this is not a correct instructions for MIPS.

∴ Joey is correct.

3. Using the optimized multiplication algorithm, **show** each step of the multiplication of 12 by 3.

Solution:

12×3

1100 (MN)
0011 (MR)

Initially:	MN:	1100	
	P:	$0000\ 0011$	
Step 1:	P:	$1100\ 0011$	$P_{LS} = P_{LS} + MN$
	P:	$\underline{0}110\ 0001$	$= 0000+1100$
			$= 1100$
Step 2:	P:	$0010\ 0001$	$P_{LS} = P_{LS} + MN$
	P:	$\underline{1}001\ 0000$	$= 0110+1100$
			$= 0010$ (Carry 1)
Step 3:	P:	$\underline{0}100\ 1000$	
Step 4:	P:	$\underline{0}010\ 0100$	

∴ Product, P = $0010\ 0100$ (36)

Summer 2023

1.

You are the most renowned computer architect in the world. Your friend asks you to find the answer to the following question for the given scenario: Consider computer A with a CPU speed of 2 GHz. The program P which has number of instruction and CPI as shown in Table 1.

Instruction	IC	CPI
addi	3	2
add	2	1
beq	1	4
bne	0	8
slt	1	4
sll	1	4
and	1	4
j	1	8

Table 1: Program P's number of instructions and CPI in Computer A architecture

1. a) Find the execution time of program P in Computer A.

Solution:

Given,

CPU Clock Rate $= 2\ GHz$

$= 2 \times 10^9\ Hz$

Now,

Clock Cycle = IC × CPI

$= 3 \times 2 + 2 \times 1 + 1 \times 4 + 0 \times 8 + 1 \times 4 + 1 \times 4 + 1 \times 4 + 1 \times 8$

$= 32$

We know,

CPU Time = Clock Cycles × Clock Time

$= \dfrac{32}{2 \times 10^9}$

$= 16 \times 10^{-9}\ s$

$= 16\ ns$

∴ Execution time program P in computer A is $16\ ns$.

1. b) Computer B with a CPU speed of 10 GHz has the execution time of the program P, 2 times faster than computer A and same instruction set architecture (ISA), calculate the average CPI of computer B.

Solution:

From 1(a),

Clock Cycle$_A$ $= 32$

Here,

$$\text{Total Instruction Count of Computer A} = 3+2+1+0+1+1+1+1 = 10$$

$$\therefore \text{Average CPI}_A = \frac{32}{10} = 3.2$$

For computer A,

$$\text{CPU Time}_A = \frac{\text{IC}_A \times \text{Average CPI}_A}{\text{Clock Rate}_A} = \frac{I \times 3.2}{2 \times 10^9} = I \times 1.6 \times 10^{-9}\ s$$

For computer B,

$$\text{CPU Time}_B = \frac{\text{IC}_A \times \text{Average CPI}_B}{\text{Clock Rate}_B} = \frac{I \times \text{Average CPI}_B}{2 \times 10^9} = I \times \text{Average CPI}_B \times 5 \times 10^{-10}\ s$$

Here,

$\text{Clock Rate}_A = 2\ GHz = 2 \times 10^9\ Hz$

$\text{Clock Rate}_B = 10\ GHz = 10 \times 10^9\ Hz$

$\text{IC}_A = \text{IC}_B = I$

$\text{Average CPI}_A = 3.2$

$\text{Average CPI}_B = ?$

According to question,

$$\frac{\text{CPU Time}_A}{\text{CPU Time}_B} = 2$$

$$\text{or, } \frac{I \times 1.6 \times 10^{-9}}{I \times \text{Average CPI}_B \times 5 \times 10^{-10}} = 2$$

$$\text{or, } \frac{1.6 \times 10^{-9}}{2 \times 5 \times 10^{-10}} = \text{Average CPI}_B$$

$$\therefore \text{Average CPI}_B = 1.6$$

∴ Average CPI of computer B is 1.6.

1. c) **As we get the total execution time of the given program for Computer A from question (a). The Arithmetic unit takes 62.5%, the logical unit takes 25%, and the branch operation takes 12.5% time of total execution. Find the improvement factor of the given program if we replace the arithmetic unit with a better Arithmetic unit, which improves total completion time by 2x.**

Solution:

From 1(a),

$$\text{Total execution time} = 16\ ns$$

$$\therefore \text{Arithmetic unit time} = (16 \times 62.5\%)\ ns = 10\ ns$$

$$\therefore \text{Logical unit time} = (16 \times 25\%)\ ns = 4\ ns$$

$$\therefore \text{Branch operation time} = (16 \times 12.5\%)\ ns = 2\ ns$$

Here,

$T_{affected} = 10\ ns$ [Arithmetic unit time]

$$T_{uneffected} = (16 - 10)\ ns$$
$$= 6\ ns$$
$$T_{improved} = \frac{16}{2}\ ns$$
$$= 8\ s$$
Improvement factor, $n = ?$

We know,
$$T_{improved} = \frac{T_{affected}}{n} + T_{unaffected}$$
or, $8 = \dfrac{10}{n} + 6$

or, $\dfrac{10}{n} = 2$

or, $\dfrac{10}{2} = n$

$\therefore n = 5$

2. Consider the following C function. Assume necessary registers.

```
2000: int find_series_sum(int n,int x){
          intres=0;
          if(n==0){
              res = x;
          }
          else{
              for(int i=1;i<=n;i++){
                  res = res + (x*5);
              }
          }
          return res;
      }
      int main(){
1000:     int s=10,x=2,r=0;
          r = find_serise_sum(s,x);
      }
```

2. a) Convert the code to the corresponding MIPS assembly instructions.

Solution:

The code has been converted into MIPS assembly instructions:

```
   1000:  addi $s0, $zero, 10
   1004:  addi $s1, $zero, 2
   1008:  addi $s2, $zero, 0
   1012:  add  $a0, $s0, $zero
   1016:  add  $a1, $s1, $zero
   1020:  jal  ~~find_series_sum~~ 2000
   1024:  add  $s2, $v0, $zero
   .
   .
find_series_sum:
```

Assuming,

s in \$s0
x in \$s1
r in \$s2

```
  2000:  addi $sp, $sp, -8
  2004:  sw   $s0, 4($sp)
  2008:  sw   $s1, 0($sp)
  2012:  addi $s0, $zero, 0
  2016:  bne  $a0, $zero, ~~else~~ 2024
  2020:  add  $s0, $a1, $zero
else:
  2024:  addi $s1, $zero, 1
loop:
  2028:  slt  $t0, $s1, $a0
  2032:  beq  $t0, $zero, ~~exit_loop~~ 2056
  2036:  sll  $t0, $a1, 2
  2040:  add  $t0, $t0, $a1
  2044:  add  $s0, $s0, $t0
  2048:  addi $s1, $s1, 1
  2052:  j    ~~loop~~ 2028
exit_loop:
  2056:  add  $v0, $s0, $zero
  2060:  sw   $s1, 0($sp)
  2064:  sw   $s0, 4($sp)
  2068:  addi $sp, $sp, 8
  2072:  jr   $ra
```

Assuming,

res in $s0
i in $s1

Changing

`for(int i=1;i<=n;i++)`
to
`for(int i=0;i<n;i++)`

2. b) Convert the first 8 lines of your assembly instructions to the corresponding machine code. No need to convert it to binary.

Solution:

The code has been converted into MIPS assembly instructions:

1.

op	rs	rt	C/A
8	0	16	10

2.

op	rs	rt	C/A
8	0	17	2

3.

op	rs	rt	C/A
8	0	18	0

4.

op	rs	rt	rd	shamt	funct
0	16	0	4	X	32

5.

op	rs	rt	rd	shamt	funct
0	17	0	5	X	32

6.

op	address
3	500

7.

op	rs	rt	rd	shamt	funct
0	2	0	18	X	32

8.

op	rs	rt	C/A
8	29	29	-8

2. c) Assume we have a new instruction type available in MIPS architecture which is S-type. Only load/store instructions can be executed using the S-type MIPS field. The structure of the Stype is given in table 2. Please find the maximum number of indexes that can be possible in an array. Explain your answer.

op	rs	rt	C/A
8 bits	8 bits	8 bits	40 bits

Table 2: S-type format for MIPS

Solution:

Given,
Structure of S-Type instruction,

op	rs	rt	C/A
8 bits	8 bits	8 bits	40 bits

Here,
Maximum value for C/A $= 2^{40} - 1$
$= 1.09951163 \times 10^{12}$

This is the maximum value that we can store in the C/A. But it is not the maximum index number because we cannot use consecutive number for array index location since each location is size of 4 byte. Therefore if we divide maximum value with 4, then we will get maximum index for array.

∴ Maximum index for array $= \dfrac{1.09951163 \times 10^{12}}{4}$
$= 2.74877907 \times 10^{11}$

∴ Maximum number of indexes that can be possible in an array is $2.74877907 \times 10^{11}$.

3. a) Assuming 4-bit architecture and using the division algorithm show each step of the division of 13 by 5.

Solution:

$13 \div 5$

1101 (DN)
0101 (DR)

Initially:	A, Q: 0000 1101 M: 0101 -M: 1011	M = 0101 1010 $\underline{+1}$ -M = 1011
Step 1:	A, Q: 0001 101_ A, Q: 1100 101$\underline{0}$ A, Q: 0001 1010	A = A-M = 0001+1011 = 1100
Step 2:	A, Q: 0011 010_ A, Q: 1110 010$\underline{0}$	A = A-M = 0011+1011 = 1110

A, Q: 0011 0100

Step 3:	A, Q: 0110 100_ A, Q: 0001 100$\underline{1}$	A = A-M = 0110+1011 = 0001
Step 4:	A, Q: 0011 001_ A, Q: 1110 001$\underline{0}$ A, Q: 0011 0010	A = A-M = 0011+1011 = 1110

∴ Reminder, A = 0011 (3)
∴ Quotient, Q = 0010 (2)

3. b) If we want to multiply 32 by 32 using the multiplication algorithm then what will be the minimum size of the product register?

Solution:

Here,

$$\begin{array}{r} 32 \\ \underline{\times\ 32} \\ 1024 \end{array}$$

Let, n bits size required for product register.

Now,

$$2^n - 1 = 1024$$
or, $2^n = 1024 + 1$
or, $n = \log_2 1025$
or, $n = 10.001$
$\therefore n \approx 11$

∴ Minimum size of the product register should be 11 bits.

Spring 2023

1. a) Alternative compiled code sequences are using same instructions as **add**, **sub** and **beq**. A table is given to show the required number of cycles per instruction (CPI) and the instruction count (IC) on each code sequence.

Find out the number of clock cycles and average CPI for all the code sequences.

Instuction	add	sub	beq
CPI	3	4	7
Code Sequences 1	240	300	500
Code Sequences 2	320	100	150

Solution:

For Code Sequence 1,

$$\text{Clock Cycle}_1 = \text{IC} \times \text{CPI}$$
$$= 3 \times 240 + 4 \times 300 + 7 \times 500$$
$$= 5420$$
$$\text{Total Instruction Count} = 240 + 300 + 500$$
$$= 1040$$
$$\therefore \text{Average CPI}_1 = \frac{5420}{1040} = 5.21$$

For Code Sequence 2,

$$\text{Clock Cycle}_2 = \text{IC} \times \text{CPI}$$
$$= 3 \times 320 + 4 \times 100 + 7 \times 150$$
$$= 2410$$
$$\text{Total Instruction Count} = 320 + 100 + 150$$
$$= 570$$
$$\therefore \text{Average CPI}_2 = \frac{2410}{570} = 4.23$$

1. b) Consider a computer running a program that requires 400 s, with 90 s spent executing FP instructions, 180 s executed L/S instructions, and 60 s spent executing branch instructions. Find out the affected and unaffected times for Amdahl's law. What is the improvement factor using Amdahl's law if we get the program completion time improved by 4x?

Solution:

Here,

$$T_{affected} = (90 + 180 + 60)\ ns$$
$$= 330\ ns$$
$$T_{uneffected} = (400 - 330)\ ns$$
$$= 70\ s$$

$$T_{improved} = \frac{400}{4}\ ns$$
$$= 100\ ns$$
Improvement factor, $n = ?$

We know,

$$T_{improved} = \frac{T_{affected}}{n} + T_{unaffected}$$
or, $100 = \frac{330}{n} + 70$

or, $\frac{330}{n} = 30$

or, $\frac{330}{30} = n$

$\therefore n = 11$

1. c) What is power wall? Discuss the necessity of multi-core processors.

Solution:

The power wall refers to the physical limitations in increasing processor speed due to excessive power consumption and heat generation. The power wall is a point where increasing processor speed becomes impractical due to the insurmountable challenges of power consumption and cooling.

To overcome the power wall, the computer industry shifted its focus from increasing clock speeds to increasing the number of cores on a single chip. The power wall necessitated a shift in processor design, leading to the development of multi-core processors as a viable solution to overcome the limitations of single-core processors performance. Multi-core processors offer several advantages such as improved performance, enhanced energy efficiency, parallel processing and scalability.

2. Consider the following C function. The starting MIPS assembly instruction is 1000. Assume necessary registers.

```
int function(int n1, int n2) {
    int i,s=1;
    for(i=n1;i<n2;i++) {
        if(arr[i]<5) {
            arr[i]=arr[i]+(s*5);
            s=s+i;
        }
        else {
            s++;
        }
    }
    return s;
}
```

2. a) Convert the code to the corresponding MIPS assembly instructions.

Solution:

The code has been rewritten below with the error correction:

```
function:
  1000:  addi $sp, $sp, -8
  1004:  sw   $s0, 4($sp)
  1008:  sw   $s1, 0($sp)
  1012:  addi $s0, $zero, 1
  1016:  add  $s1, $a0, $zero
loop :
  1020:  slt  $t0, $s1, $a1
  1024:  beq  $t0, $zero, ~~exit_loop~~ 1084
  1028:  sll  $t0, $s1, 2
  1032:  add  $t1, $s2, $t0
  1036:  lw   $t2, 0($t1)
  1040:  slti $t0, $t2, 5
  1044:  beq  $t0, $zero, else
  1048:  sll  $t0, $s0, 2
  1052:  add  $t3, $t0, $s0
  1056:  add  $t0, $t2, $t3
  1060:  sw   $t0, 0($t1)
  1064:  add  $s0, $s0, $s1
  1068:  j ~~loop_end~~ 1076
else:
  1072:  addi $s0, $s0, 1
loop_end:
  1076:  add  $s1, $s1, 1
  1080:  j    ~~loop~~ 1020
exit_loop:
  1084:  add  $v0, $s0, $zero
  1088:  sw   $s1, 0($sp)
  1092:  sw   $s0, 4($sp)
  1096:  addi $sp, $sp, 8
  1100:  jr   $ra
```

Assuming,

n1 in $a0
n2 in $a1
s in $s0
i in $s1
arr[] in $s2

[Assuming arr[] was declared outside of function before]

2. b) **Convert the first 10 lines of your assembly instructions to the corresponding machine code. No need to convert it to binary..**

Solution:

The code has been converted into MIPS assembly instructions:

1.

op	rs	rt	C/A
8	29	29	-8

2.

op	rs	rt	C/A
43	29	16	4

3.

op	rs	rt	C/A
43	29	17	0

4.

op	rs	rt	C/A
8	0	17	1

5.

op	rs	rt	rd	shamt	funct
0	4	0	17	X	32

6.

op	rs	rt	rd	shamt	funct
0	17	5	8	X	42

7.

op	rs	rt	C/A
4	0	8	271

8.

op	rs	rt	rd	shamt	funct
0	X	17	8	2	0

9.

op	rs	rt	rd	shamt	funct
0	18	8	9	X	32

10.

op	rs	rt	C/A
35	9	10	0

2. c) Assume we have a new instruction type available in MIPS architecture which is K-type. Only jump instruction can be executed using the K-type MIPS field. Structure of the Ktype is given below. Please find the maximum jump address. Explain your answer.

op	rs	rt	C/A
12 bits	10 bits	10 bits	32 bits

Solution:

Given,
Structure of K-Type instruction,

op	rs	rt	C/A
12 bits	10 bits	10 bits	32 bits

Here,
Maximum value for C/A $= 2^{32} - 1$
$= 4294967295$

This is the maximum value that we can store in the C/A. But it is not the jump address because we store jump address in C/A by dividing it by 4. Therefore if we multiply maximum value with 4, then we will get maximum jump address.

∴ Maximum index for array $= 4294967295 \times 4$
$= 17179869180$

∴ Maximum jump address is 17179869180.

3. a) Assuming 4-bit architecture and using the division algorithm show each step of the division of 11 by 6.

Solution:

$11 \div 6$

1011 (DN)

0110 (DR)

Initially: A, Q: 0000 1011
M: 0110
-M: 1010

M = 0110
1001
$\underline{\quad +1}$
-M = 1010

Step 1: A, Q: 0001 011_
A, Q: 1011 011$\underline{0}$
A, Q: 0001 0110

A = A-M
= 0001+1010
= 1011

Step 2: A, Q: 0010 110_
A, Q: 1100 110$\underline{0}$
A, Q: 0010 1100

A = A-M
= 0010+1010
= 1100

Step 3: A, Q: 0101 100_
A, Q: 1111 100$\underline{0}$
A, Q: 0101 1000

A = A-M
= 0101+1010
= 1111

Step 4: A, Q: 1011 000_
A, Q: 0101 000$\underline{1}$

A = A-M
= 1011+1010
= 0101

∴ Reminder, A = 0101 (5)
∴ Quotient, Q = 0001 (1)

3. b) Optimized multiplication is better than the normal multiplication algorithm. Why? Explain.

Solution:

Given,
Structure of K-Type instruction,

op	rs	rt	C/A
12 bits	10 bits	10 bits	32 bits

Here,
Maximum value for C/A $= 2^{32} - 1$
$= 4294967295$

This is the maximum value that we can store in the C/A. But it is not the jump address because we store jump address in C/A by dividing it by 4. Therefore if we multiply maximum value with 4, then we will get maximum jump address.

∴ Maximum index for array $= 4294967295 \times 4$
$= 17179869180$

∴ Maximum jump address is 17179869180.