RAMAIAH UNIVERSITY OF APPLIED SCIENCES

ASSIGNMENT

Course Code	19CSC205A
Course Name	Microprocessor & Assembly Language.
Programme	B. Tech
Department	Computer Science & Engineering
Faculty	Faculty of Engineering Technology

Name of the Student	SUBHENDU MAJI
Reg. No	18ETCS002121
Semester/Year	3RD / 2019
Course Leader/s	Supriya M.S.

Declaration Sheet

Student Name	SUBHENDU MAJI		
Reg. No	18ETCS002121		
Programme	B. Tech	Semester/Year	3rd / 2019
Course Code	19CSC205A		
Course Title	Microprocessor & Assembly Language		
Course Date		To	
Course Leader	Supriya M.S.		

Declaration

The assignment submitted herewith is a result of my own investigations and that I have conformed to the guidelines against plagiarism as laid out in the Student Handbook. All sections of the text and results, which have been obtained from other sources, are fully referenced. I understand that cheating and plagiarism constitute a breach of University regulations and will be dealt with accordingly.

Signature of the Student		Date	
Submission date stamp (by Examination & Assessment Section)			

Signature of the Course Leader and date	Signature of the Reviewer and date

Faculty of Engineering & Technology			
Ramaiah University of Applied Sciences			
Department	Computer Science and Engineering	**Programme**	B. Tech.
Semester/Batch	3rd /2018		
Course Code	19CSC205A	**Course Title**	Microprocessors and Assembly Programming
Course Leader	P.Padma Priya Dharishini , Supriya M.S.		

Assignment - 1						
Name of Student		Register No				
Sections		**Marking Scheme**	Max Marks	First Examiner Marks	Second Examiner Marks
Part-A	A1.1	Assembly Language Program	3		
	A1.2	Clock cycle time, Execution time of sequence recognizer, CPI	3		
	A1.3	AMAT	2		
	A1.4	Comparison of Execution time	2		
		Max Marks	**10**		

Course Marks Tabulation				
Component- CET B Assignment	**First Examiner**	**Remarks**	**Second Examiner**	**Remarks**
Marks (out of 10)				
Signature of First Examiner				Signature of Second Examiner

Question No. 1

Solution to Question No. 1:

A1.1 Assembly Language Program

```
# sequence recognizer
.section .data
ar:
    .int 0,0,0,0,0,0,0,0,0,0
ar_len:
    .int 10
.section .bss

.section .text

.globl _start
# function for system exit code
_ret:
    movq    $60, %rax               # sys_exit
    movq    $0, %rdi                # exit code
    syscall

# driver function
_start:
    movl $0b0010110110,%esi
    movl ar_len,%edx
    subl $1,%edx

    loop:
        movl %esi,%edi
        andl $0b1111,%edi
        sarl $1,%esi

        cmp $0,%edx
        je _end

        cmp $0b1011,%edi
        je equal
        jne not_equal

    equal:
        movl $1,ar(,%edx,4)
        subl $1,%edx
        jmp loop

    not_equal:
        movl $0,ar(,%edx,4)
        subl $1,%edx
        jmp loop
    _end:


    # for converting array to binary
    movl $0,%ecx
    movl $0,%eax

    loop1:
        orl ar(,%eax,4),%ecx
        sall $1,%ecx
        addl $1,%eax
        cmp $9,%eax
        jne loop1


    syscall
    call _ret            # exit
```

Figure 1 ASM code

```
subhendu@subhendu:/mnt/d/RUAS/sem 03/MP lab/assignment$ as -gstabs assign.s -o assign.o
assign.s: Assembler messages:
assign.s: Warning: end of file in comment; newline inserted
subhendu@subhendu:/mnt/d/RUAS/sem 03/MP lab/assignment$ ld assign.o -o assign
subhendu@subhendu:/mnt/d/RUAS/sem 03/MP lab/assignment$ gdb assign
GNU gdb (Ubuntu 8.1-0ubuntu3) 8.1.0.20180409-git
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from assign...done.
(gdb)
```

Figure 2 Making and linking file

Output when searching sequence **1011** when binary no. is **0010110110**:

```
(gdb) p (int[10])ar
$1 = {0, 0, 0, 0, 0, 1, 0, 0, 1, 0}
(gdb) p /t $ecx
$2 = 10010
(gdb)
```

Figure 3 output for input 0010110110

Output when searching sequence **1011** when binary no. is **101101011010**:

```
(gdb) p /t $ecx
$1 = 100001000
(gdb) p (int[12])ar
$2 = {0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0}
(gdb)
```

Figure 4 output for input 101101011010

Output when searching sequence **1100** and binary no. is **110011001**:

```
(gdb) p /t $ecx
$1 = 100010
(gdb) p (int[9])ar
$2 = {0, 0, 0, 1, 0, 0, 0, 1, 0}
(gdb)
```

Figure 5 output for input 110011001

A1.2 Clock cycle time, Execution time of sequence recognizer, CPI

Table 1 Clock Cycle and CPI Calculation

Line No.		source	Destination	No. of Clock Cycles for each instruction	No. of clock cycles repeated per instruction	no. of clock cycles
20		Immediate	Register	1	1	1
21		Memory	Register	7	1	7
22		Immediate	Register	1	1	1
	loop:					
25		Register	Register	1	10	10
26		Immediate	Register	1	10	10
27		Immediate	Register	1	10	10
29		cmp		5	10	50
30		je		5	10	50
32		cmp		5	10	50
33		je		5	10	50
34		jne		5	5	25
	equal:					
37		Immediate	Memory	1	5	5
38		Immediate	Register	1	5	5
39		jmp		5	5	25
	not_equal:					
42		Immediate	Memory	1	5	5
43		Immediate	Register	1	5	5
44		jmp		5	5	25
49		Immediate	Register	1	1	1
50		Immediate	Register	1	1	1
	loop1					
53		Memory	Register	7	10	70
54		Immediate	Register	1	10	10
55		Immediate	Register	1	10	10
56		cmp		5	10	50
57		jne		5	10	50

We know, $Clock\ Cycle\ Time = \frac{1}{clock\ rate}$

$$\boldsymbol{Clock\ cycle\ time} = \frac{1}{1.60 * 10^9} = \mathbf{0.625\ ns}$$

Refer to Fig. 6 for Clock Rate.

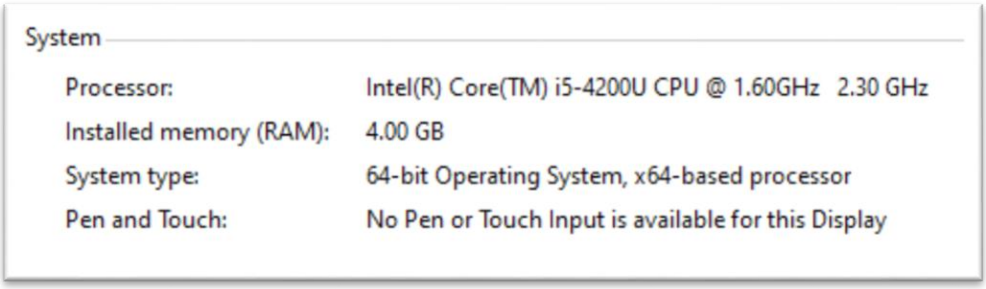

Figure 6 Clock Rate = 1.60 GHz

Note. This Calculation is only valid for specific input, i.e. **0b0010110110**. Refer to Fig. 1 for the source code.

From Table 1, The calculations are as follows:

$$\boldsymbol{Instruction\ Count} = 1 + 1 + 1 + 10 * (1 + 1 + 1 + 1 + 1) + 10 * (1 + 1) + 5 * 1 + 5 * (1 + 1 + 1) + 5 * (1 + 1 + 1) + 1 + 1 + 10 * (1 + 1 + 1 + 1 + 1) = \mathbf{160}$$

$$\boldsymbol{Total\ no.of\ clock\ Cycles} = 1 + 7 + 1 + 10 * (1 + 1 + 1 + 5 + 5) + 10 * (5 + 5) + 5 * 5 + 5 * (1 + 1 + 5) + 1 + 1 + 10 * (7 + 1 + 1 + 5 + 5) = \mathbf{526}$$

We know,

$$CPI = \frac{CPU\ Clock\ Cycles}{Instruction\ Count}$$

$$\boldsymbol{CPI} = \frac{526}{160} = \mathbf{3.28\ CPI}$$

We know,

$$Execution\ Time = No.of\ \ ClockCycles * Clock\ Cycle\ \ Time$$

$$Execution\ Time = \frac{No.of\ Clock\ Cycles}{Clock\ Rate}$$

$$\boldsymbol{Execution\ Time} = 526 * 0.625 = \mathbf{328.75\ ns}$$

A1.3 AMAT

We know, $\boldsymbol{AMAT} = \boldsymbol{hit\ time} + (\boldsymbol{miss\ rate} * \boldsymbol{miss\ penalty})$

Two-level cache **AMAT formula**:

$\boldsymbol{AMAT} = \boldsymbol{Hit\ Time_{L1}} + (\boldsymbol{Miss\ Rate_{L1}} * \boldsymbol{Miss\ Penalty_{L1}})$ ------**equation 1**

Where,

$Miss\ Rate_{L1} = 1 - Hit\ Rate_{L1}$

$Miss\ Penalty_{L1} = Hit\ Time_{L2} + (Miss\ Rate_{L2} * Miss\ Penalty_{L2})$

Given,

Parameters	L1 cache	L2 cache
Associativity	2-way set Associative	4-way set Associative
Block size	32 bytes	64 bytes
Cache size	512 Kb	1Gb
Hit rate	40%	60%
Miss Penalty	-	30
Hit time	4 ns	20 ns

$$\boldsymbol{Miss\ Rate_{L1}} = 1 - Hit\ Rate_{L1}$$
$$= 1 - 0.4 = \boldsymbol{0.6} = \boldsymbol{60}\%$$

$$\boldsymbol{Miss\ Rate_{L2}} = 1 - Hit\ Rate_{L2}$$
$$= 1 - 0.6 = \boldsymbol{0.4} = \boldsymbol{40}\%$$

$$\boldsymbol{Miss\ Penalty_{L1}} = Hit\ Time_{L2} + (Miss\ Rate_{L2} * Miss\ Penalty_{L2})$$
$$= 20\ ns + 0.4 * 30$$
$$= 20 + 12$$
$$= \boldsymbol{32\ ns}$$

Substituting the values in equation 1,

$$\boldsymbol{AMAT} = \boldsymbol{Hit\ Time_{L1}} + (\boldsymbol{Miss\ Rate_{L1}} * \boldsymbol{Miss\ Penalty_{L1}})$$
$$AMAT = 4 + (0.6 * 32)$$
$$= 4 + 19.2$$
$$\boldsymbol{AMAT} = \boldsymbol{23.2\ ns}$$

A1.4 Comparison of Execution time

The actual execution time can be shown by **time** command, which is as follows:

```
real    0m0.006s
user    0m0.000s
sys     0m0.000s
subhendu@subhendu:/mnt/d/RUAS/sem 03/MP lab/assignment$ time ./assign

real    0m0.007s
user    0m0.000s
sys     0m0.000s
subhendu@subhendu:/mnt/d/RUAS/sem 03/MP lab/assignment$ time ./assign

real    0m0.008s
user    0m0.016s
sys     0m0.000s
subhendu@subhendu:/mnt/d/RUAS/sem 03/MP lab/assignment$
```

Figure 7 when using time command

Here, we are getting 0.00 sec , even after running program for twice or thrice.
And, as we can see, the inbuilt '**time**' command is not so accurate. It can show up to milliseconds only. it is not capable of showing the time in Nano-seconds.

Hence, I have used a '**shell script**' to find accurate execution time using '**date**' command.
Refer to Fig. 5 for Source Code of the Shell script.

```
calc.sh
#!/bin/bash
start=`date +%s%N`
for i in {1..1000}; do ./assign; done;
end=`date +%s%N`
echo "Execution Time : $(((end-start)/1000)) nanoseconds"
```

Figure 8 script file code for running the program 1000 times and getting time in Nano seconds

Basically, here in the Shell Script, I am storing a starting time and then running the program. then when the program ends, I am storing that time.
We can get an average Execution Time by **'(end-start)/1000'**
The Shell Script is running the program for 1000 times. So, I can achieve an average accurate value up to some extent. As running the program for once or twice may not give accurate value.

```
subhendu@subhendu:/mnt/d/RUAS/sem 03/MP lab/assignment$ ./calc.sh
Execution Time : 5227178 nanoseconds
subhendu@subhendu:/mnt/d/RUAS/sem 03/MP lab/assignment$
```

Figure 9 output of running script file

Theoretically we are getting Execution time to be **328.75 ns** but practically, we are getting an average Execution time to be **5227178 ns**.

As we can see, there is a large difference between manually calculated Execution time and Execution time given by computer. This is because manually calculated time is calculated under ideal condition, but practically there a lot of background process happening which is slowing down the Execution.
An another reason can be that the program is running on WSL (windows Subsystem for LINUX). It can be faster if the main OS of the system is UBUNTU.

- Refer to https://github.com/subhendu17620/RUAS/tree/master/sem%2003/MP%20lab/assignment for the ASM code
- https://cs.stackexchange.com/questions/95659/amat-calculation