Assignment 03

Ali Raza
BCS F19 M 513

Computer Organization
AND
Assembly Language

SECTION - I

1) Intel Gen.s till Now:-

Up till now, intel has introduced 11 generations of processors. These are :

- Nehalem (1st)
- Sandy Bridge (2nd)
- Ivy Bridge (3rd)
- Haswell (4th)
- Broadwell (5th)
- Skylake (6th)
- Kaby Lake (7th)
- Kaby Lake R (8th)
- Coffe Lake (9th)
- Cannon Lake / Ice lake (10th)
- Tiger Lake (11th)

2) F.P. instructions in Intel 8086:-

Intel designed a seperate floating point chip named 8087 in 1978 to perform/run floating point instructions on

Intel 8086 processor. This chip contains eight 80-bit data registers organized as a stack and intel named these registers as ST0, ST1 ST2 ---- ST7.

3.

	4004	8080	8086	80386	80686
Data-Bus	4-bit	8-bit	16-bit	32-bit	64-bit
Address-Bus	12-bit	16-bit	20-bit	32-bit	64-bit
Size of Cache					
Clock rate	400-800KHz	2MHz	5-10MHz	16-33MHz	3.8+ GHz

4.

a) Intrupts: Intel 8008

b) Protected Memory: 80286

c) DMA Controler: 80286

d) Memory paging & Virtual memory: 80386

e) Super scalar Architect- 80586 - P5

f) SSE: 80686 - P6

g) out of order execution, register

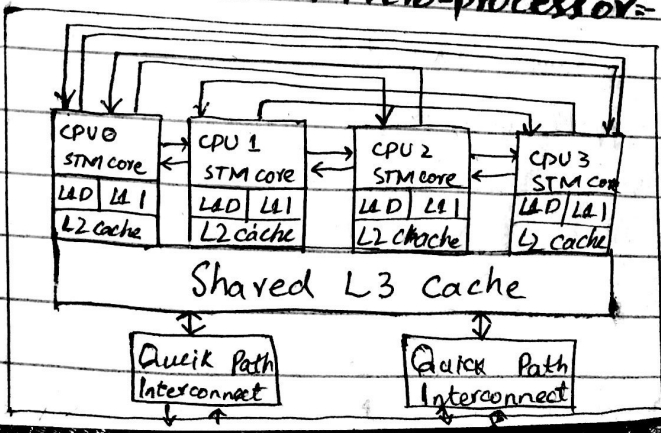
running, Branch prediction:-
Intel 80686 - P6

5. Pantium III & Pantium Iv

Pantium-III was an Intel brand/brand based on P6 architecture which introduced a new SIMD technology called Streaming SIMD Extension (SSE).

Pantium-II wan an Intel brand/~~brand~~ model based on p6 architecture, having a clock support of upto 3.8+ GHz and support of hyper-threading technology.

6. Intel Cor Micro-processor:-



7 Core Solo:

Intel core solo microprocessor family is a family of single-core (mobile) processor based on improved pantium M micro-architecture. It has 32 KB level 1 caches, 2 MB level 2 cache support of SSE & SSE2 instructions.

Dual Core:

Intel Core duo is a family of dual-core mobile) processor based on ~~embe~~ enhanced mobile pantium M micro architecture. These are integrated on two improved pantium M cores on the die. Each core have 32KB L1 caches, Both cores share the same 2MB L2 cache. It also support SSE & SSE2 instructions.

8. **i3:**

No turbo boost, slower clock speed, 2 cores - 4 threads, HT enabled.

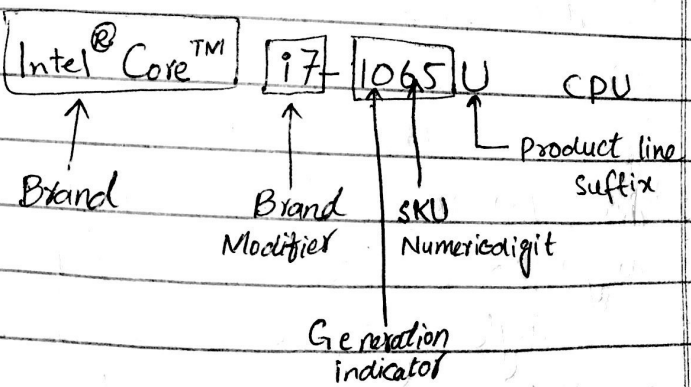
i5:

Comes with turbo boost. 4-cores-4-threads, HT disabled.

i7:

Enabled turbo boost, 4-cores 8-threads, more cache more clock speed, HT enabled.

9. **Latest intel processor naming Convention:-**



10.

$IC = 1.0 \times 10^7$

$CPI = 0.4 \times 3 + 0.35 \times 5 + 0.25 \times 2 = 3.45$

$Clock\ Time = 1/(3 \times 10^9) = 3.33 \times 10^{-10}$

Exe time:-

$= IC \times CPI \times Clock\ Time$

$= 1.0 \times 10^7 \times 3.45 \times 3.33 \times 10^{-10}$

$= 0.011499$ sec.

11: Single cycle & Multi cycle:-

A single cycle CPU executes each instruction in one cycle. Whereas multi-cycle CPU executes each instruction multiple cycles.

The clock cycle of a single cycle CPU must be large enough to execute the comple instruction.

Single cycle CPU is easy to implement as compared to multi-cycle CPU.

12: Advantages of Single S. CPU:

The only advantage of single cycle CPU is that it is not so complex to easy to implement.

Disadvantage:

Disadvantages are that single cycle CPU must operate on the speed of slowest instruction. Clock cycle will accordingly be larged.

13. Why Multi-Cycle

Multicycle CPU takes variable amount of time to execute instruction, depending on their size. It breaks complex instructions into small, easy simple and executes them faster as compared to Single Cycle CPU.

14:

Stages = 6

instruction = 80

Now for clock cycle:

K stage pipeline:-

n^{th} instruction will complete in $K+(n-1)$ cycles.

So

$6+(80-1) = 85$ clock cycle.

15. Even & Uneven pipeline:-

In even pipeline each stage takes equal time to execute, but in unevent pipeline each stage takes variable clock time.

16. Intermediate Registers:-

Intermediate registers are used to hold the intermediate output between two stages. They are also called buffer or latch. All intermediat registers along with stages are

controlled by a common clock.

17. Classes of pipeline Hazards:-

Class:-

- Structural Hazards
- Data Hazards
- Control Hazards

↳ Structural Hazards occures when multiple instructions are trying to access same resource (memory or cache).

i.e, in a same cycle if instruction is fetched and memory is being accessed.

↳ Data Hazards occure when an instruction depends on result of a prior instruction still in pipeline.

```
ADD R1, R2, R3
SUB R4, R5, R1
```

↳ Control Hazards occure when instruction that changes the PC is being execute.

```
CMP R1, R2
JEQ loop / 1048
```

18. Out of order execution.

Control Hazard occures in case of out of order execution.

Solutions:

- Flush the pipeline
- Delayed Branch
- Dynamic Branch prediction
  - ↳ 1 or 2 bit prediction
  - ↳ Correlating prediction
  - ↳ Tournament prediction.

19.

In the absence of out of order execution, everything goes on as normal. But in case of branching, pipeline is flushed and new instructions are created, according to out-of-order execution.

clock cycle	1	2	3	4	5	6	7	8	9
CMP R1, R2	IF	ID	EX	MEM	WB				
JEQ 1048		IF	ID	EX	MEM	WB			
Ins # n						IF	ID	EX	
Ins # n+1							IF	ID	
Ins # n+2								IF	

20. **RISC:**

↳ Simple and less number of instructions, easy to decode.

↳ Fixed instruction set

↳ Instructions takes one cycle to execute.

CISC:

↳ Complex and more number of instructions, difficult to decode.

↳ Instructions have variable size

↳ Instructions take varying amount of clock cycles.

SECTION -II

1. Segmented memory model

In segmented memory model, memory is divided into groups of independent segments referenced by pointers located in special CPU registers.

Intel used segmented memory

model in 8086 CPU because they wanted to run all assembly programs running on 8080 to run on 8086 as well. 8086's memory of 1 Mib was divided into 64 KB segmented, so 8080 program could be loaded in it and execute successfully.

2- Real, protected & long Modes:

Real mode of 80386 was meant for backward compatibility which allowed programs of 8086 to run on 80386. 80386 had memory of 4 GiB and 8086 had 1 MiB of memory. So in real mode only first 1 Mib of 80386 could be used.

Real mode of x86-64 works just like 8086 and support, real mode flat model and real mode segmented model.

protected mode of 80386 uses full 4GB, 32-bit support much larger data-structures than real mode.

Protected mode of x86-64 works just like 80386.

Long mode of x86-64 is true 64-bit mode, all instructions registers are 64-bit wide.

3.

8086 segmented memory model was a short-term success but on the long way it was catastrophic. Programs who do not have to use segmented memory had to use it forcefully. If large data structures were needed, programmers had to use it in chunks. So it was not a success to introduced segmented memory model.

4.

	8080	8086	80386	x86-64
Address Bus	16-bit	20-bit	32-bit	64-bit
Data Bus	8-bit	16-bit	32-bit	64-bit

5.

8086 memory was divided into 64KiB segments for backward compatibility. The memory size of 8080 was 64 KiB so to run assembly programs of 8080, 8086 memory of 1MiB was divided into 64KiB segments.

6

A5B8 : D4E6

$A5B8 * 10_{(16)} + D4E6$

physical address: A5B80 + D4E6 = B0006

7

In 8086 processors the 20 bits address can be stored in 16-bit segment register because the right most bits are set to zero and need not to stored.

8:

CF: set if last addition results carry
PF: set if number of 1s is even.
AF: used for BCD Arithmetic
ZF: set if result is zero.
SF: set if result is negative.

9:

ES: Extra segment Base Address
SS: Stack segment base Address
CS: Code segment base Address
DS: Data segment " "
DI: Extra segment offset "
BD: Base of stack
SP: Top of stack.

IP: code segment offset address
SI: Data segment Base address

10.

x86-64 uses 48-bits because, 64-bits logical address can address a huge amount of memory that can never be fully used. So 48-bits of logical address is used and 48 to 63 bits are unused.

11

Pros:

↳ It teaches us working of hardware.

↳ Reverse engineering

↳ Optimizing program for size & speed.

↳ To write device drivers, OS Kernal

Cons:

- Portability is lost.
- Hard to read and write code
- debugging is difficult.

12. NASM:

NASM is an assembler and disassembler for x86 architecture. It can be used to write 16-bit, 32-bit and 64-bit programs. NASM is one of the most popular assemblers for linux.

13:

X86.64 architecture is a multi-cycle CPU, every operation happens in multiple cycles. For example to fetch instructions, machine instructions can be of different size than performing other operations.

Q.14 Linker & loader.

↳ Linker Links or attaches one or more object files, some library code into some executable file.

↳ Loader Loads executable code in memory & executes the program.

15.

Source code file

↓

$ nasm -felf64 fileName.nasm

↓

object code file

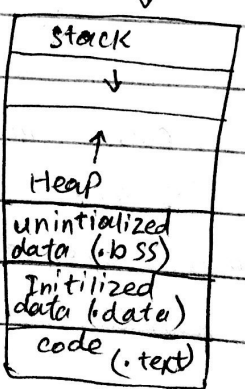
$ ld fileName.o -o myExe.out

↓

Executable file

$ ./myExe.out

↓



SECTION - III

1. Types of Statements:-

i) Assembly Inst.

These instructions are actually converted into machine code when executed tell processor what to do. ie MOV, ADD, SUB.

ii) Pseudo Inst.:-

These are not read x86 instructions but normally used in real instruction field. ie (DW, RESB, EQU).

iii) Assemble Directives:

Statements for Assemble to do something. i.e. (SECTION, EXTERN)

2. MOV inst

MOV instruction is used to move data from one storage space to other.

MOV dest, source

FORMS:-

There are five forms:

→ MOV register, immediate

MOV rax, 1

→ MOV register, register

MOV rdx, rsi

→ MOV register, Memory

MOV rdi, msg

→ MOV memory, register

MOV msg, rdi

→ MOV memory, Immediate

MOV msg, 1

3. Operands:-

Operands for assembly instructions are registers, memory and intermediate.

MOV rax, 1

MOV rdx, 1

4. _start or main

_start or main define the starting point or entry point.

They indicate when to start the execution of assembly program from ld. expects program entry point to be _start and gcc expect main.

5. Sections:-

3 sections:-

i) SECTION .data (initilized data)

ii) SECTION .bss (un-initilized data memory)

iii) SECTION .text (contains actual code)

6. System Call:-

System call is a controlled entry point into the OS code, which allows a process to request the OS to perform a privileged operation. With syscall, an ID is given which request OS

to perform a service based on that ID.

7.

```
SECTION .bss
        myString resb 17
```

8.

```
SECTION .data
        EXIT_STATUS equ 54
SECTION .bss
        input resb 17
SECTION .text
        global _start
        _start
        mov rax, 0
        mov rdi, 0
        mov rsi, input
        mov rdx, 17
        syscall
        move rax, 60
        move rdi, EXIT_STATUS
        syscall
```

9.

In section (.data)

SECTION .data

EXIT_status equ 49

only this change is required.

10.

syscall has 6 operands or arguments in x86-64 architecture. So for these 6 operands, 6 registers are used.

rdi, rsi, rdx, r10, r8, r9

11.

TASKS performed by debuggers are:

1. stop a program on a certain condition.
2. Examin what has happend when program is stopped.
3. Changes thing while program is runing, to se behaviour in different scenarios.

12. why (gdb):-

A programmer uses gdb for :-

→ Run time analysis
→ Disassembly
→ Analyze program flow
→ Reverse engineering
→ Cracking binaries

13.

We can use shell commands inside (gdb) by using '!' sign befor command.

For example:-

(gdb) ! clear

14:

$ info scope copystring

15:

$ list 1, 16

or $ l 1, 16

or $ file: 1, file: 16

16.

set variable var1 = 10

17.

set disassembly-flavor intel

18.

The variables can be used within gdb to keep hold of a certain value, and use it later. These are entirely within gdb and have no effect on execution of program.

(gdb) set $i = 15

(gdb) print $i

$1 = 15

19. (gdb) a.c : line#

(gdb) break 3

(gdb) a.c : Func Name

(gdb) break _start

(gdb) break 0x2ff5bbcc2100

20

Using "call" command.

(gdb) call myadd(25,34)