COAL LOGICAL REASONING QS

Q) Consider the following code. What is the value of AX after execution of the following instructions?

```
MOV AX, 10H
MOV CX, 0AH
L1:
INC AX
DEC CX
LOOPNE L1
```

Explanation:
CX is decremented two times , one time by the DEC CX instruction and other by LOOPNE instruction. The loop runs 5 times and so AX is incremented 5 times.
10h+5h = 15h
The loop terminates when either CX is 0 or ZF =1 is set.
The loop continues when CX != 0 and ZF = 0 is not set.

Ans: AX = 15h

Q) How many bytes are allocated as a result of the following data definition directive:
X1 BYTE 20 DUP(30 DUP('COAL'))

Explanation:
Each character takes one byte space.
30 * 4= 120.
120 * 20 = 2400

Ans: 2400 BYTES

Q) How do assemblers and linkers work together?
The assembly code is converted into machine language (object code) by assembler, Linker then links all the necessary libraries and predefined code with the object file to make it executable.

Q) Differentiate between the following Assembly Language instructions:
MOV EBX, OFFSET X1
MOV EBX, X1
The first instruction copies the base address of X1 in EBX, second instruction copies the contents.

COAL LOGICAL REASONING QS

Q) Elaborate the difference between MOVZX and MOVSX instructions through an example.
MOVZX is unsigned extension of source operand, MOVSX extends the sign.
e.g.
MOV AL, 9C
MOVZX CX, AL ; CX = 009Ch
MOVSX DX, AL ; DX = FF9Ch

Q) Write equivalent x86 assembly instructions for the following operation:
POP EBX

MOV EBX, [ESP]
ADD ESP, 4

Q) Write a single x86 instruction to replace each of the following two instructions:

LEA EBX, X1
MOV EBX, OFFSET X1
LODSB
MOV AL, [ESI]

Q) When do you typically use the CBW and CWD instructions?
With 8bits, and 16bits signed divisions respectively.

Q) Write the x86 assembly PROTOYPE for following sample function:
void sample(int length, char ch, int arr[]);
sample PROTO, length: DWORD, ch: BYTE, arrPtr: PTR DWORD

Q) Write a valid x86 assembly INVOKE instruction for the following sample function:
int sample(int arr[], int num, int* x)
INVOKE sample, ADDR arr32, mem32, ADDR var32

Q) Write a valid x86 assembly function signature using PROC for the following sample function:
void sample(char ch, int num, char chArr[])
sample PROC, ch: BYTE, num: DWORD, chPTR: PTR BYTE

Q) What will happen, if immediately upon entering a subroutine you execute a "POP" instruction?
The subroutine will lose its return address.

Q) Consider the following code snippet. What is the value of AX after execution of the following instructions?

```
MOV AX, 90F8h
MOV CX, 08h
L1: SHRD AX, CX, 1
JC L2
LOOP L1
L2: RET
```

Explanation:

```
Initially: AX 1001  0000  1111  1000
     08h    CX 0000   0000  0000  1000
```

The LSB of AX goes into CF which is 0
The LSB of source which rightnow is 0 moves to MSB of AX.

```
           AX 0100  1000  0111  1100
           CX decrements due to LOOP L1 instruction
07h        CX 0000  0000  0000 0111
```

The LSB of AX goes into CF which is 0
The LSB of source which rightnow is 1 moves to MSB of AX.

```
           AX 1010  0100  0011  1110
           CX decrements due to LOOP L1 instruction
06h        CX 0000  0000  0000 0110
```

The LSB of AX goes into CF which is 0
The LSB of source which rightnow is 0 moves to MSB of AX.

```
            AX  0101  0010  0001  1111
           CX decrements due to LOOP L1 instruction
05h        CX 0000  0000  0000 0101
```

The LSB of AX goes into CF which is 1.
The LSB of source which rightnow is 1 moves to MSB of AX.

```
            AX  1010  1001  0000  1111
```

Since CF = 1, so it moves to L2 which says return.
Hence final values of AX is A90Fh and CX is 05h

Ans: CX = 05h, AX = A90Fh

COAL LOGICAL REASONING QS

Q) What will be the value of AX after the following code is executed?

```
MOV BL, 12
MOV AX, 98
DIV BL
```

Explanation:

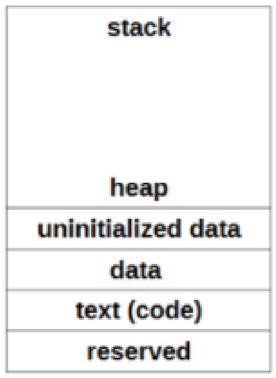
98/12 = 8.167

Quotient = 8, remainder = 2

Quotient goes into AL and remainder into AH.

Ans: AX = 0208h

Draw and briefly discuss the general memory layout of a program in MIPS architecture.



Q) How address, data, and control busses are used during the instruction execution?

They are used to access memory (data operands or instructions). Address bus generates a 20 bit address and the control bus facilitates reading or writing control. Data is transferred to/from memory on data bus.

Q) How do you differentiate a data label from a code label?

Data labels are names of variables, pointing somewhere in the data segment whereas code label specifies some point in code part.

Data label:

```
Var1 BYTE 20h
Count DWORD 100
```

Code label:

```
L1:
… code body
Loop L1
```

Q) Explain why memory access takes more machine cycles than register access?

Because Register are located within the CPU, no addressing is involved hence no need to fetch data through busses, access delay is minimal which is not the case when accessing memory.

COAL LOGICAL REASONING QS

Q) What is the difference between CALL and JMP instructions in context of branching?
CALL pushes return address on the stack in order to return to the point after the call, JMP simply braches to destination.

Q) "Stack parameters are more convenient than register parameters", prove with an example.
Registers are limited in number, hence it is more convenient to use stack parameters.

Q) Why is sign extension necessary before signed division (IDIV)?
In order to preserve the SIGN of operand.
Example:

```
.data
byteVal SBYTE -48 ; D0 hexadecimal
.code
mov al,byteVal ; lower half of dividend
cbw ; extend AL into AH
mov bl,+5 ; divisor
idiv bl ; AL = -9, AH = -3
```

Notice the difference of doing sign extension on answer.

```
.data
byteVal SBYTE -48 ; D0 hexadecimal
.code
mov al,byteVal ; lower half of dividend
mov bl,+5 ; divisor
idiv bl ; AL = 41, AH = 3
```

Q) With the help of an example, clarify when does one-operand IMUL set the Carry Flag and the Overflow Flag?
IMUL sets the Carry Flag and Overflow flag when the upper half of the product is NOT the sign extension of the lower half.
E.g.

```
mov al,48
mov bl,4 ; 48*4 = 192 in decimal(it exceeds 128) hence OF.
imul bl ; AX = 00C0h, OF = 1
```

COAL LOGICAL REASONING QS

Q) How LOOPE is different from the LOOP instruction? Provide an example.
LOOPE checks for ZF == 1 and CX != 0 , whereas LOOP only checks for CX != 0.

LOOPNE/LOOPNZ VS LOOPE/LOOPZ
LOOPE L1
– ECX ← ECX – 1
– if ECX > 0 and ZF=1, jump to label L1 in this case.
Example: scanning an array for the first element that does not match a given value.

Whenever you want to terminate loop when value is found (ZF = 1). So use loopne/loopnz.

LOOPNE L1
– ECX ← ECX – 1
– if ECX > 0 and ZF=0, jump to label L1 in this case.
Example: scanning an array for the first element that matches a given value.

• The following code finds the first positive value in an array:
```
.data
array SWORD -3,-6,-1,-10,10,30,40,4
sentinel SWORD 0
.code
mov esi , OFFSET array
mov ecx , LENGTHOF array
next:
test WORD PTR [esi] , 8000h ; test sign bit
pushfd ; push flags on stack
add esi , TYPE array
popfd ; pop flags from stack
loopnz next ; continue loop
jnz quit ; none found
sub esi,TYPE array ; ESI points to value
Quit:
```
Loop terminates as soon as first positive value is found (ZF = 1).

Locate the first nonzero value in the array. If none is found, let ESI point to the sentinel value
```
.data
array SWORD 50 DUP (?)
sentinel SWORD 0
.code
mov esi , OFFSET array
```

COAL LOGICAL REASONING QS

```
mov ecx , LENGTHOF array
next:
cmp WORD PTR [esi] , 0 ; check for zero
pushfd ; push flags on stack
add esi , TYPE array
popfd ; pop flags from stack
loopnz next ; continue loop
jnz quit ; none found
sub esi,TYPE array ; ESI points to value
quit:
```

Loop terminates as soon as first nonzero value is found (ZF = 1).

Q) When does a divide overflow occur at machine level? Give an example to illustrate.
Divide overflow occurs at the machine level when the result of division operation cannot be represented in the destination register due to its size limitations. This happens when:

1) Dividend is the minimum possible signed integer(most negative number).
2) The divisor is -1.

Range of a 32-bit signed integer:
-2^31 , 2^31 -1
Minimum value: -2^31 = -2,147,483,648
Maximum value: 2^31 = 2,147,483,647
Performing the division: -2,147,483,648 / -1 = 2,147,483,648
This exceeds the maximum representable value.
Code Example:

```
mov eax, -2,147,483,648d
mov ecx, -1d
idiv ecx
```

Q) What are the two ways of passing arguments to a procedure?

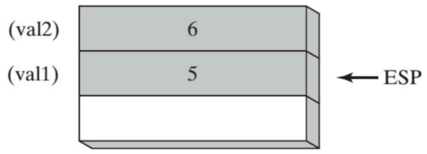
1) Pass by value.
2) Pass by reference.

By value ;

•When an argument is passed by value, a copy of the value is pushed on the stack.

E.g. in C++:

```
int sum = AddTwo( val1, val2 );
.data
val1 DWORD 5
val2 DWORD 6
.code
push val2
push val1
call AddTwo
```



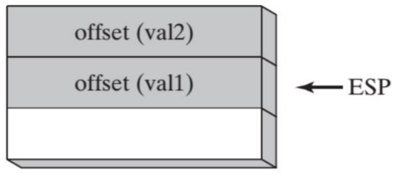
By reference:

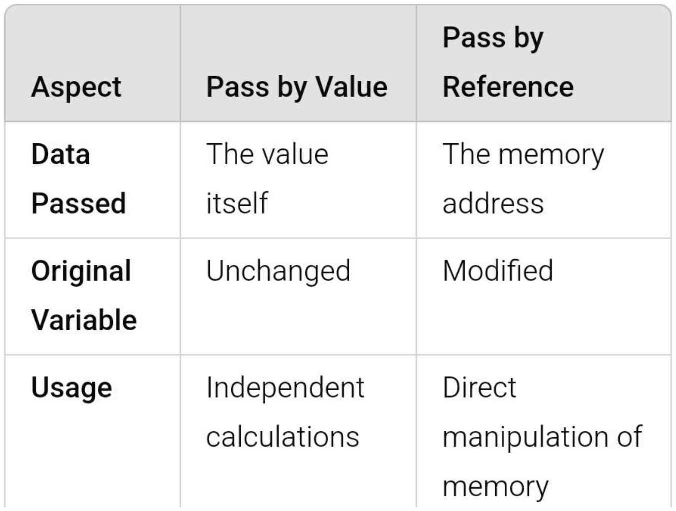
•An argument passed by reference consists of the address (offset) of an object

```
push OFFSET val2
push OFFSET val1
call Swap
```

In C++:

```
Swap( &val1, &val2 );
```





Aspect	Pass by Value	Pass by Reference
Data Passed	The value itself	The memory address
Original Variable	Unchanged	Modified
Usage	Independent calculations	Direct manipulation of memory

COAL LOGICAL REASONING QS

CALL VS RET

The CALL instruction calls a procedure by directing the processor to begin execution at a new memory location

- ➔ Pushes offset of next (instruction after call) on the stack
- ➔ Copies the address of the called procedure into EIP

The RET instruction returns from a procedure

- ➔ Pops top of stack into EIP

PUSHAD, PUSHA:

•The PUSHAD instruction pushes all of the 32-bit general-purpose registers on the stack in the following order: EAX, ECX, EDX, EBX, ESP (value before executing PUSHAD), EBP, ESI, and EDI.
•The POPAD instruction pops the same registers off the stack in reverse order.

•Similarly, the PUSHA instruction, pushes the 16-bit general-purpose registers (AX, CX, DX, BX, SP, BP, SI, DI) on the stack in the order listed. The POPA instruction pops the same registers in reverse.

RET VS RET 8:

1. RET (Return Instruction)
It pops the return address from the stack into the EIP (Instruction Pointer) register.
The execution continues from the address stored in EIP.
The stack pointer (ESP) is adjusted automatically to remove the return address.

2. RET 8 (Return with Stack Cleanup)
Pops the return address into the EIP register.
Additionally, it increments the stack pointer (ESP) by the value specified (e.g., 8).

ENTER:

•The ENTER instruction performs three operations:
1. Pushes EBP on the stack (push ebp)
2. Sets EBP to the base of the stack frame (mov ebp, esp)
3. Reserves space for local variables (sub esp,numbytes)

ENTER numbytes, nestinglevel

ENTER 0,0 ->procedure with no local variables.
ENTER 8, 0 ->Reserves 8 bytes of stack space for local variables.

COAL LOGICAL REASONING QS

LEAVE:
•The LEAVE instruction terminates the stack frame for a procedure.
• It reverses the action of a previous ENTER instruction by restoring ESP and EBP to the values they were assigned when the procedure was called.

USES operator:
•USES tells the assembler to do two things:
1. First, generate PUSH instructions that save the registers on the stack at the beginning of the procedure.
2. Second, generate POP instructions that restore the register values at the end of the procedure.
USES esi ecx
Equivalent code for the above statement:
```
push esi
push ecx
.
.
.
pop ecx
pop esi
```

LOCAL VS ENTER: LOCAL declares local variables by name, with it's type for e.g: BYTE, DWORD etc. ENTER reserves space for unnamed local variables.

BIT TEST(BT) Instruction:
Copies bit n from an operand into the Carry flag
Syntax: BT bitBase, n
bitBase may be r/m16 or r/m32
n may be r16, r32, or imm8
Example: jump to label L1 if bit 9 is set in the AX
register:
```
bt AX,9
; CF = bit 9
jc L1
; jump if Carry
```
Mov instruction is the only instruction that doesn't change flag registers.

CMP and SUB Instruction:
Both operations affects flag registers.
Cmp doesn't store the result.
In subtract, the value of destination is changed.

COAL LOGICAL REASONING QS

Example:
mov al, 20h
sub al, 5h
Result al= 20-5 = 15h
mov al, 20h
cmp al, 5h
It subtracts 20-5 but doesn't update the value of al.

AND and TEST Instruction:
AND:
Destination is updated.
Flag registers are updated.
TEST:
Destination isn't updated.
Flag registers are updated.

MOV and SHLD:
MOV dest, source
Destination and source must be of same size, e.g: both of 8bits/16bits.
Used to move all bits.
SHLD dest, source, CL
Destination and source must be of same size, CL/imm8 can be different.
Used to move particular bits only.

LAHF and SAHF: The following flags are copied: Sign , Zero, Auxiliary, Carry and Parity.

```
.data
saveflags BYTE  ?
.code
lahf   ; load flags into AH
mov saveflags , ah   ; save them in a variable
mov saveflags , ah  ; mov back to ah
sahf  ; save them/ copy them in the Flags Register
```

Q) Is it possible for a NEG instruction to set the Overflow flag? Explain in one sentence.
Yes its possible for NEG instruction to set the Overflow flag such as when negating -2^31 the minimum value for 32-bit signed integer.

Q) How do you declare a two-dimensional 4x4 array of bytes in assembly language? Give the code.

```
.data
array BYTE 4 DUP (4 DUP(0))
```

COAL LOGICAL REASONING QS

Q) How CLD and STD instructions are used with the string primitive instruction?
CLD (Clear Direction Flag) causes string operations to move forward (incrementing the index registers).
STD (Set Direction Flag) causes string operations to move backward (decrementing the index registers).

Q) What is meant by a one-to-many and one-to-one relationship in context of x86 assembly. Use a coding example to support your answer.
A single statement in C++ expands into multiple assembly language or machine instructions. (one-to-many relationship).
Each assembly language instruction corresponds to a single machine-language instruction. (one-to-one relationship).

Q) Explain the Direct, In-Direct and Indexed Modes of accessing a memory in x86 assemblies using an example instruction.
A direct memory operand is a named reference to storage in memory:

```
.data
var1 BYTE 10h
.code
mov al,var1
```

An indirect operand holds the address of a variable, usually an array or string. It can be

```
derefereYTE 10h,20h,30h
.code
mov esi,OFFSET val1
mov al,[esi]
```

An indexed operand adds a constant to a register to generate an effective address.

```
.data
arrayW WORD 1000h,2000h,3000h
.code
mov esi,0
mov ax,[arrayW + esi]
```

Q) Briefly explain the role of Virtual Machine(s) in execution of a high level language code.
The high level codenced (just like a pointer).
.data
val1 B is compiled to Java byte code and then the JVM executes the code on the hardware platform that is running.

COAL LOGICAL REASONING QS

Q) Why is Assembly language not portable?
Assembly language is not portable because it is designed for a specific processor family.

Q) Explain the difference(s) between Real and Protected modes of x86 processors.
Real Mode
only 1 MB of memory can be accessed from 0 to FFFFF.
Program can access any part of main memory.
MS-DOS runs in real-address mode.
Protected Mode
Each program can access a maximum of 4GB of memory.
OS assign memory to each running program.
Program are prevalent from accessing each other memory.
Windows NT, 2000, XP and Linux used Protected mode.

Q) What is the role of memory segmentation in x86 Intel processors regarding mismatch in register size and address bus size?
Segmentation is the process in which the main memory of the computer is logically divided into different segments and each segment has its own base address. It is basically used to enhance the speed of execution of the computer system. It allows to extend the address ability of the processor, i.e. segmentation allows the use of 16 bit registers to give an addressing capability of 1 Megabytes. Without segmentation, it would require 20 bit registers.

Q) How many clock cycles are required in order to read a single value from main memory upon CPU request?
This task requires four steps and each step requires at least one clock cycle to execute.

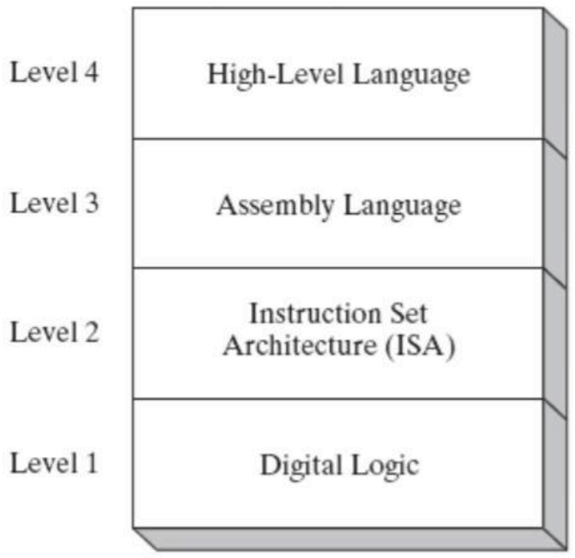
Reading a single value from memory involves four separate steps:

1. Place the address of the value you want to read on the address bus.
2. Assert (change the value of) the processor's RD (read) pin.
3. Wait one clock cycle for the memory chips to respond.
4. Copy the data from the data bus into the destination operand.

Q) Hardware view point signed vs unsigned integers.
All CPU instructions operate exactly the same on signed and unsigned integers.
The CPU cannot distinguish between signed and unsigned integers.
YOU, the programmer, are solely responsible for using the correct data type with each instruction.

COAL LOGICAL REASONING QS

•Virtual machine is a software program that emulates the functions of some other physical or virtual computer.
•The virtual machine VM1, can execute commands written in language L1.
• The virtual machine VM0 can execute commands written in language L0.
Specific Machine Levels:



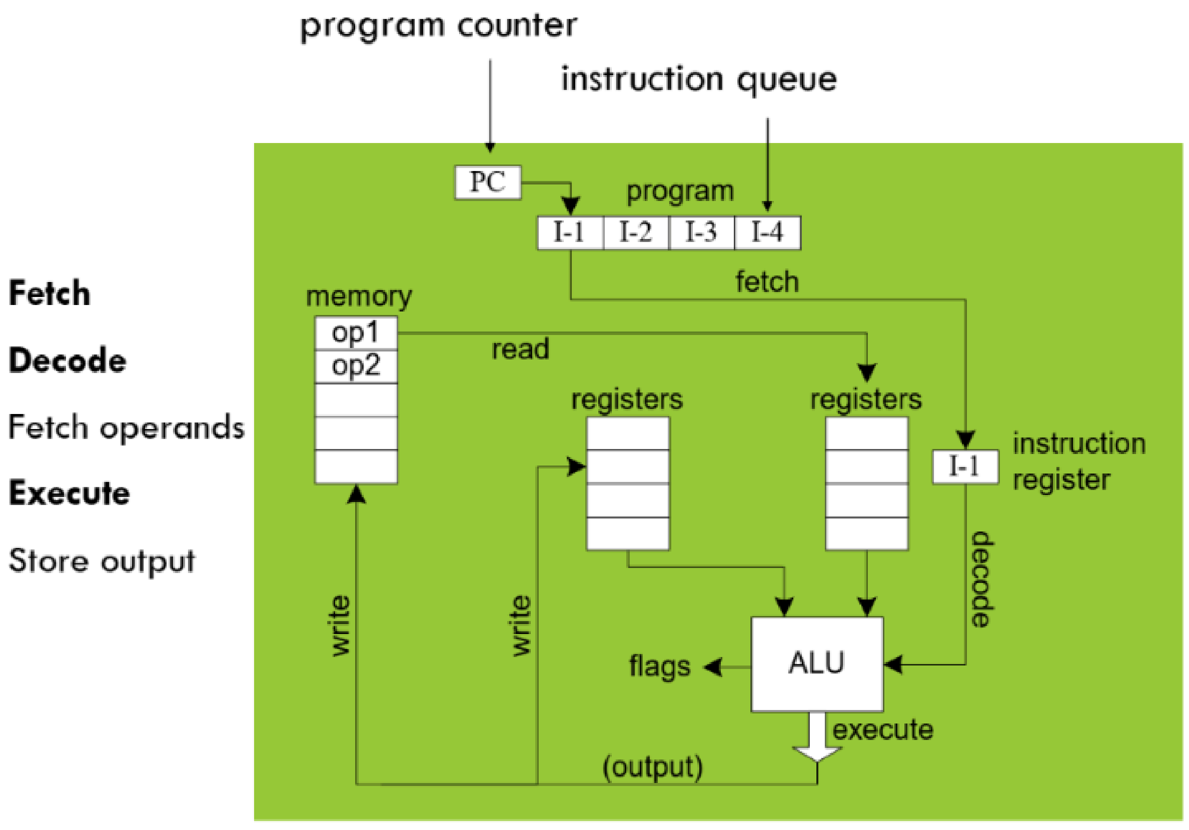
Level 1 Digital Logic
• CPU, constructed from digital logic gates
• System bus
• Memory
• Implemented using bipolar transistors.

Level 2 ISA:
• Also known as conventional machine language
• Executed by Level 1 (Digital Logic)

Level 3 Assembly language:
• Instruction mnemonics that have a one-to-one correspondence to machine language
• Programs are translated into Instruction Set Architecture Level - machine language (Level 2)

Level 4 High level language:
• Application-oriented languages
– C++, Java, Pascal, Visual Basic . . .
• Programs compile into assembly language (Level 3)

FETCH-DECODE- EXECUTE CYCLE:



INSTRUCTION EXECUTION CYCLE

Steps:

Fetch: The program counter (PC) points to the next instruction, which is fetched into the instruction queue. Advance the IP (Instruction Pointer).

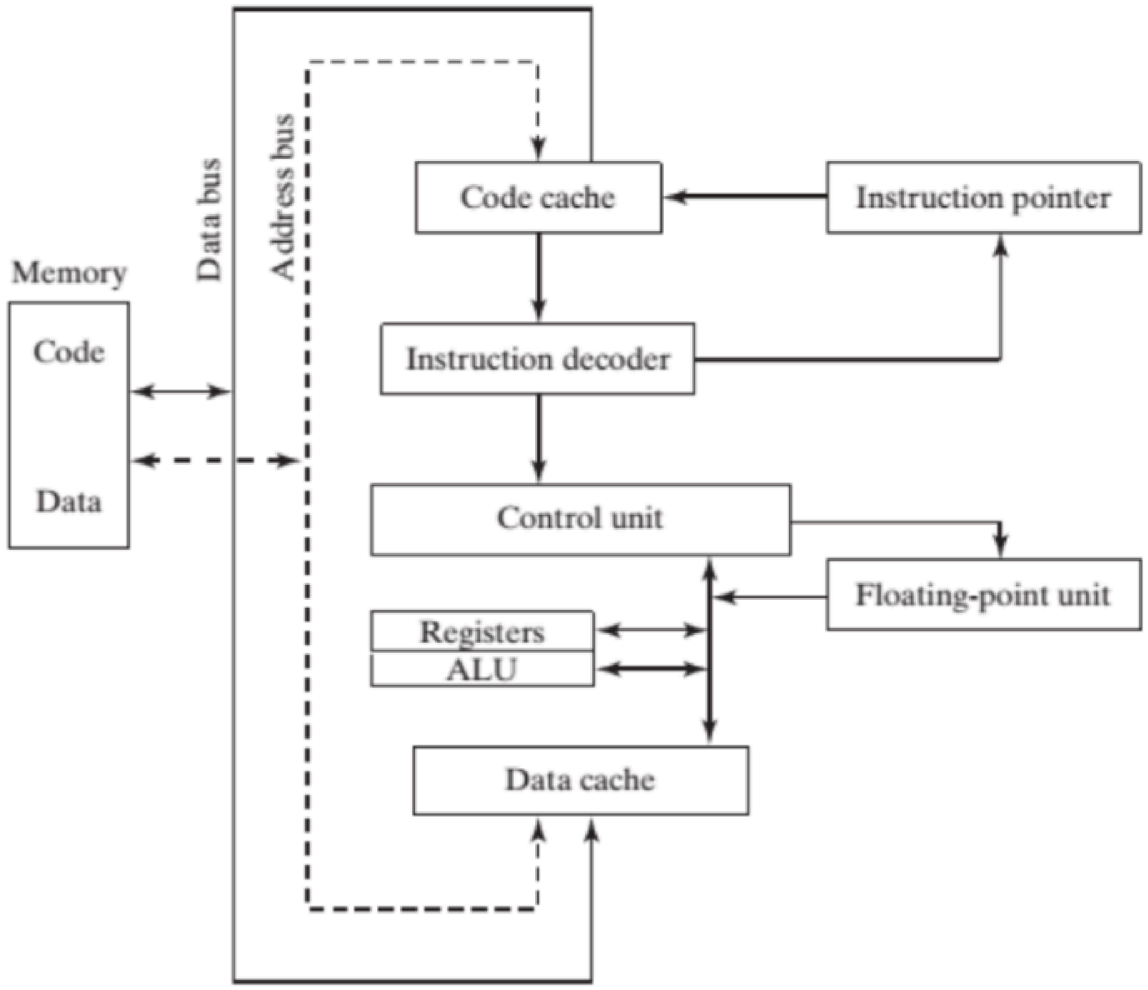
Decode: The instruction is moved to the instruction register, decoded, and operands are identified.

Fetch Operands: Operand data is read from memory or registers as required by the instruction.

Execute: The ALU performs the operation using the operands, updating flags and producing the result.

Store Output: The result is written back to memory or registers for future use.

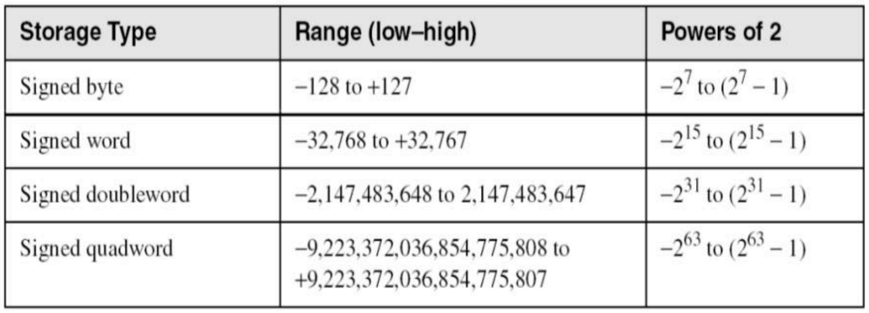
COAL LOGICAL REASONING QS



1. Memory stores both code (instructions) and data.
2. Bus is a pathway for transferring data between different parts of CPU.
3. Instruction pointer points to location in memory of next instruction to be executed.
4. Code cache speeds up the execution by reducing time it takes to retrieve information from main memory.
5. Instruction decoder translates 0101011 signals into the signals that Control Unit can understand.
6. Control Unit coordinates with ALU and floating point unit.
7. Registers temporarily hold data that's being processed.
8. ALU performs arithmetic and logical operations on data.
9. Data cache stores frequently used data for faster access.
10. After data is executed , it goes to the output device for display (depending on instruction).

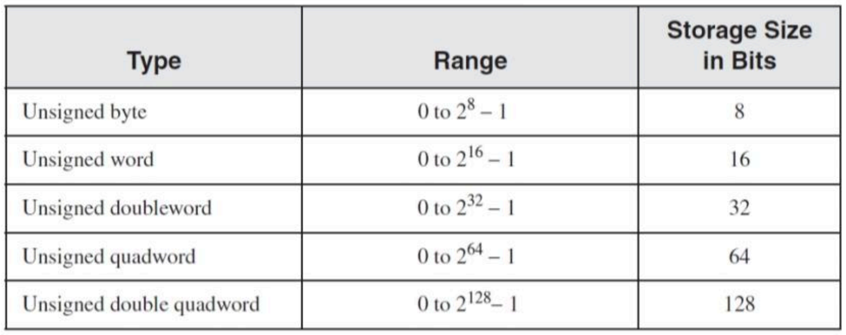
COAL LOGICAL REASONING QS

Data storage:



Storage Type	Range (low–high)	Powers of 2
Signed byte	−128 to +127	-2^7 to $(2^7 - 1)$
Signed word	−32,768 to +32,767	-2^{15} to $(2^{15} - 1)$
Signed doubleword	−2,147,483,648 to 2,147,483,647	-2^{31} to $(2^{31} - 1)$
Signed quadword	−9,223,372,036,854,775,808 to +9,223,372,036,854,775,807	-2^{63} to $(2^{63} - 1)$

Table 1-4 Ranges and Sizes of Unsigned Integer Types.



Type	Range	Storage Size in Bits
Unsigned byte	0 to $2^8 - 1$	8
Unsigned word	0 to $2^{16} - 1$	16
Unsigned doubleword	0 to $2^{32} - 1$	32
Unsigned quadword	0 to $2^{64} - 1$	64
Unsigned double quadword	0 to $2^{128} - 1$	128

Current Location Counter:

• The symbol $, is the current location counter, that returns offset of current location:
• This statement declares a variable named selfPtr and initializes it with the variable's offset value.
• As $ operator returns the offset associated with the current program statement, it can be used to calculate the size of array/string
• In the following example, ListSize is calculated by subtracting the offset of list from the current location counter ($):
list BYTE 10,20,30,40
ListSize = ($ - list)

Arrays of Words and DoubleWords
• When calculating the number of elements in an array containing values other than bytes, you should always divide the total array size (in bytes) by the size of the individual array elements.

```
list  WORD  1000h,2000h,3000h,4000h
ListSize = ($ – list) / 2
```

```
list  DWORD  10000000h,20000000h,30000000h,40000000h
ListSize = ($ –list) / 4
```

COAL LOGICAL REASONING QS

Symbolic Constants: The equal-sign directive associates a symbol name with an integer expression.
The syntax is: name = expression
Example: Count = 50
EQU directive:
name EQU expression
name EQU symbol
name EQU <text>
Examples:
1) PI EQU <3.1416>
2) pressKey EQU <"Press any key to continue...",0>
.data
prompt BYTE pressKey
3) matrix1 EQU 10 * 10
    matrix2 EQU <10 * 10>
.data
M1 WORD matrix1
M2 WORD matrix2
M1 WORD 100
M2 WORD 10 * 10

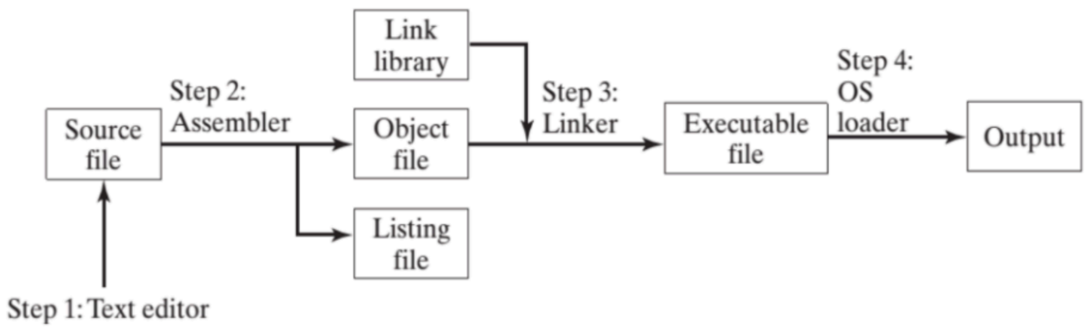
Instruction: Instruction is a statement that becomes executable when a program is assembled.
• Basic syntax
[label:] mnemonic [operands] [; comment]
Instruction Format Examples
• No operands
stc ; set Carry flag
• One operand
inc eax ; register
inc myByte ; memory
• Two operands
add ebx, ecx ; register, register
sub myByte, 25 ; memory, constant
add eax, 36 * 25 ; register, constant-expression.

Directive: commands to the assembler that directs the assembly process. E.g: .data, .stack, , .code etc.
Identifier: Variable names, e.g: _arr, arr2 etc.
Reserved words: Special meaning words in MASM, e.g: mnemonics (Add, sub, mul etc), Register names(AX, CX or BX etc), Directives, attributes .

COAL LOGICAL REASONING QS

ASSEMBLY-LINK-EXECUTE CYCLE:

Assembler is a utility program that converts source code programs from assembly language into an object file, a machine language translation of the program.
– Optionally a Listing file is also produced.
– We'll use MASM as our assembler.
• The linker reads the object file and checks to see if the program contains any calls to procedures in a link library.
– The linker copies any required procedures from the link library, combines them with the object file, and produces the executable file.
– Microsoft 16-bit linker is LINK.EXE and 32-bit is Linker LINK32.EXE.
• OS Loader: A program that loads executable files into memory, and branches the CPU to the program's starting address, (may initialize some registers (e.g. IP)) and the program begins to execute.
• Debugger is a utility program, that lets you step through a program while it's running and examine registers and memory.



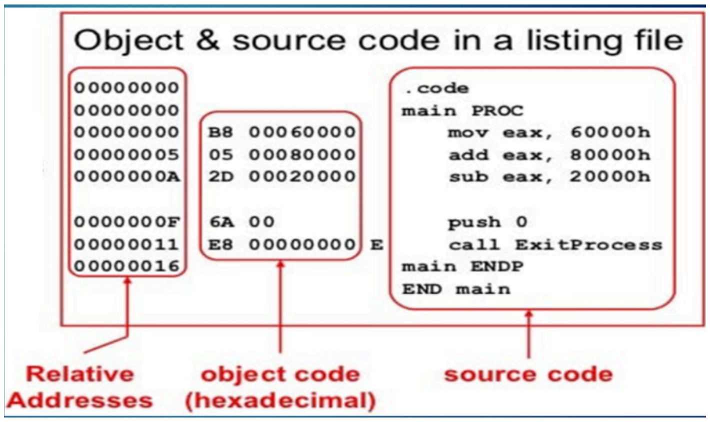
Clock: Clock synchronizes the internal operation with other system components. Clock is used to trigger events, measure the time of single operation and synchronize.

LOADING AND EXECUTING A PROGRAM:

1. The operating system (OS) searches for the program's filename in the current disk directory.
2. If it cannot find the name there, it searches a predetermined list of directories (called paths) for the filename.
3. If the OS fails to find the program filename, it issues an error message.
4. If the program file is found, the OS retrieves basic information about the program's file from the disk directory,including the file size and its physical location on the disk drive.
5. The OS determines the next available location in memory and loads the program file into memory.
6. It allocates a block of memory to the program and enters information about the program's size and location into a table (sometimes called a descriptor table).

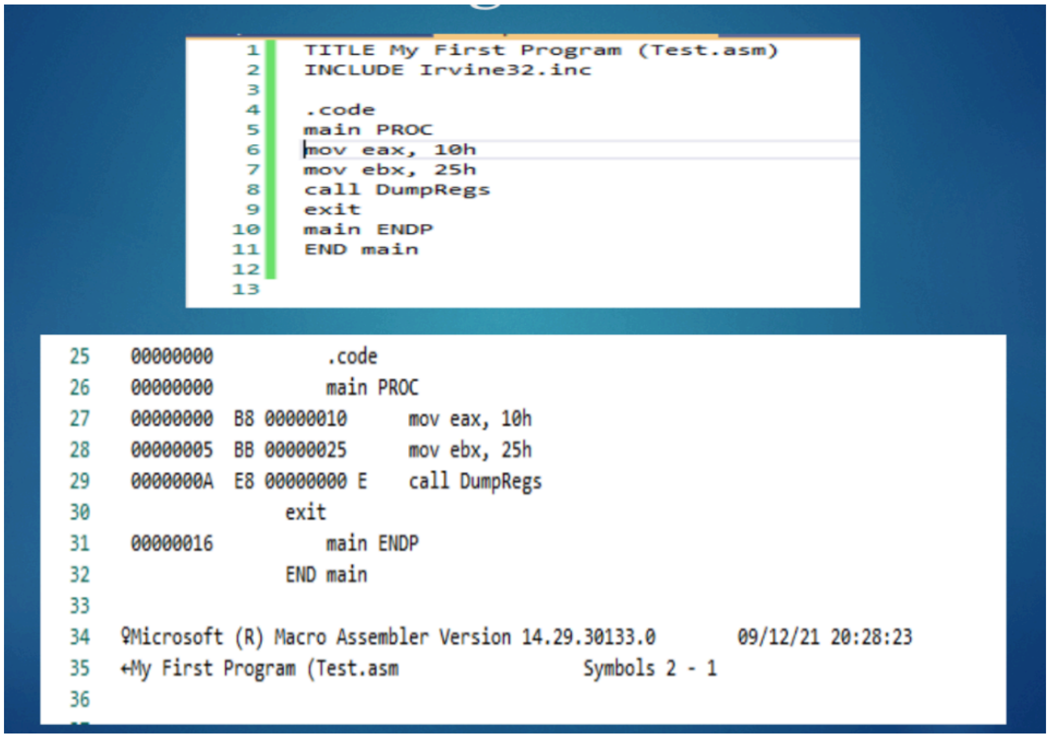
7. Additionally, the OS adjust the values of pointers within the program so they contain addresses of program data.
8. The OS begins execution of the program's first machine instruction (its entry point).
9. As soon as the program begins running, it is called a process.
10. The OS assigns the process an identification number (process ID), which is used to keep track of it while running.
11. It is the OS's job to track the execution of the process and to respond to requests for system resources.
12. Examples of resources are memory, disk files, and input-output devices.
13. When the process ends, it is removed from memory.

LISTING FILE:



Object & source code in a listing file

```
00000000                     .code
00000000                     main PROC
00000000  B8 00060000           mov eax, 60000h
00000005  05 00080000           add eax, 80000h
0000000A  2D 00020000           sub eax, 20000h

0000000F  6A 00                 push 0
00000011  E8 00000000 E         call ExitProcess
00000016                     main ENDP
                             END main
```



```
TITLE My First Program (Test.asm)
INCLUDE Irvine32.inc

.code
main PROC
mov eax, 10h
mov ebx, 25h
call DumpRegs
exit
main ENDP
END main
```

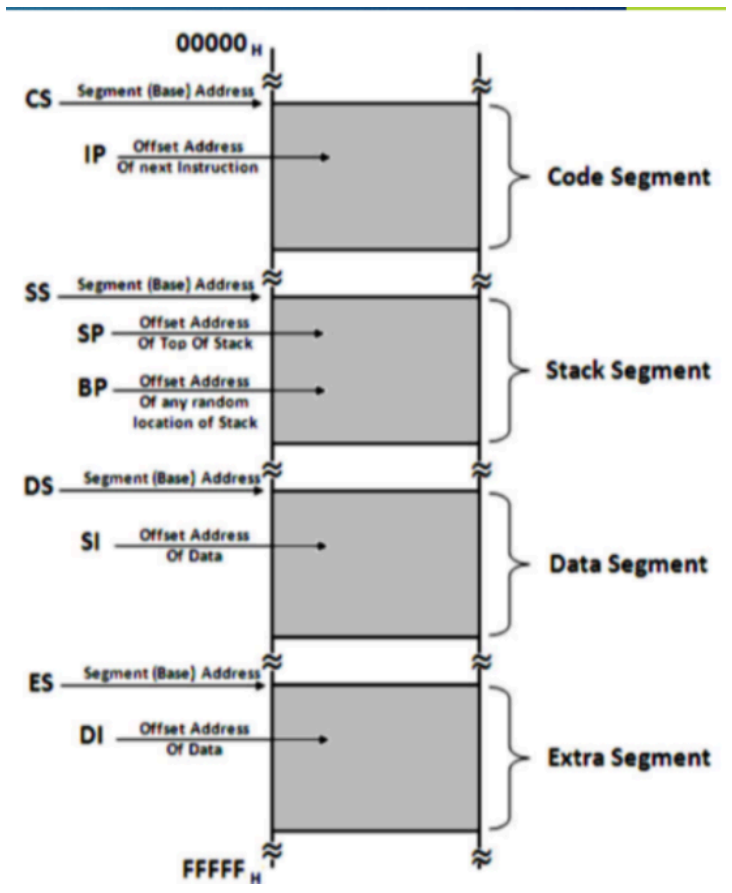
```
00000000                  .code
00000000                  main PROC
00000000  B8 00000010        mov eax, 10h
00000005  BB 00000025        mov ebx, 25h
0000000A  E8 00000000 E      call DumpRegs
                          exit
00000016                  main ENDP
                          END main

♀Microsoft (R) Macro Assembler Version 14.29.30133.0       09/12/21 20:28:23
←My First Program (Test.asm                    Symbols 2 - 1
```

COAL LOGICAL REASONING QS

MEMORY:

Seg : Offset

Physical Address = Seg * 10h + OFFSET



Q) 12AB : 025F

Segment = 12AB

Offset = 025F

12AB * 10h + 025F

12AB0 + 025F

```
 1 2 A B 0
   0 2  5  F
 1 2 D 0 F
```

Ans: 12D0Fh

COAL LOGICAL REASONING QS

FLAGS:
Zero flag: The Zero flag is set when the result of an operation produces zero in the destination operand.
A flag is set when it equals 1.
A flag is clear when it equals 0.
Example:

```
mov cx,1
sub cx,1  ; CX = 0, ZF = 1
mov ax,0FFFFh
inc ax  ; AX = 0, ZF = 1
inc ax  ; AX = 1, ZF = 0
```

Sign Flag: The Sign flag is set when the destination operand is negative.
The flag is clear when the destination is positive.
When the most significant bit is 1 (not when carry 1 is generated).
Example:

```
mov al,0
sub al,1 ; AL = 11111111b, SF = 1
add al,2 ; AL = 00000001b, SF = 0
```

Overflow and Carry Flag : Hardware viewpoint
How the ADD instruction affects OF and CF:
CF = (carry out of the MSB)
OF = CF XOR MSB

How the SUB instruction affects OF and CF:
CF = INVERT (carry out of the MSB)
negate the source and add it to the destination
OF = CF XOR MSB

Carry Flag:
The Carry flag is set when the result of an operation generates an unsigned value that is out of range (too big or too small for the destination operand).
Example:

```
mov al,0FFh
add al,1 ; CF = 1, AL = 00
; Try to go below zero:
mov al,0
sub al,1  ; CF = 1, AL = FF
```

COAL LOGICAL REASONING QS

Overflow Flag: The Overflow flag is set when the signed result of an operation is invalid or out of range.

```
; Example 1
mov al,+127
add al,1 ; OF = 1, AL = ??
; Example 2
mov al,7Fh ; OF = 1, AL = 80h
add al,1
```

RULE OF THUMB:
When adding two integers, remember that the
Overflow flag is only set when . . .

- ➔ Two positive operands are added and their sum is negative.
- ➔ Two negative operands are added and their sum is positive.

What will be the values of the Overflow flag?

```
mov al,80h
add al,92h  ; OF = 1

mov al,-2
add al,+127 ; OF = 0
```

What will be the values of the given flags after each operation?

```
mov al,-128
neg al ; CF = 1, OF = 1

mov ax,8000h
add ax,2 ; CF =0 ,OF =0

mov ax,0
sub ax,2 ; CF =1,  OF =0

mov al,-5
sub al,+125 ; OF =1
```

Parity Flag:
Parity- is set if the least-significant byte in the result contains an even number of 1 bits. It used to verify memory integrity.

COAL LOGICAL REASONING QS

Q) Compute the answers:

```
mov bx, 0A69Bh
movzx eax, bx       ;EAX = 0000A69Bh
movzx edx, bl       ;EDX = 0000009Bh
movzx cx, bl        ;CX = 009Bh
```

```
mov bx, 0A69Bh
movsx eax, bx       ;EAX = FFFFA69Bh
movsx edx, bl       ;EDX = FFFFFF9Bh
movsx cx, bl        ;CX = FF9Bh
```

```
mov ax, -128
shl eax, 16
sar eax, 16
```
EAX = FFFFFF80h

```
mov ax, 234Bh
mov dx, 7654h
shrd ax, dx, 4
```
4 bits means one hexadecimal digit.
Last hexadecimal digit which is 4 in dx moves to ax.
AX = 4234h

```
mov al,0D4h
rcl al,1
stc
mov bl, 0D4h
rcr bl, 3
```
AL = A8h
BL = 3Ah

COAL LOGICAL REASONING QS

ALIGN directive:

syntax : ALIGN bound , bound can be 1,2,4 or 16

If bound is 2, the next variable is aligned on an even numbered address.

If bound is 4, the next address is a multiple of 4.

```
bval BYTE ?        ;00404000
ALIGN 2
wVal WORD ?        ;00404002
Bval2 BYTE ?       ;00404004
ALIGN 4
dval DWORD ?       ;00404008
dval2 DWORD ?      ;0040400C
```

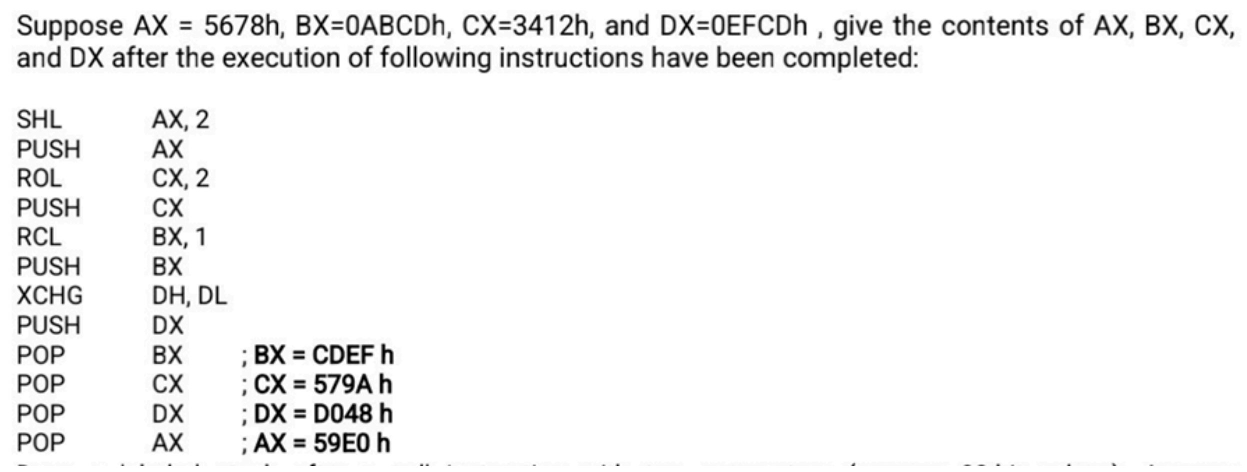
dval would have been at offset of 00404005, but ALIGN 4 moved it at offset 00404008.

Why align data?

Because the CPU can process data stored at even numbered addresses more quickly than at odd numbered addresses.

SHIFT/ROTATE:

•Shift and Rotate instructions affect the overflow and carry flags.



Suppose AX = 5678h, BX=0ABCDh, CX=3412h, and DX=0EFCDh , give the contents of AX, BX, CX, and DX after the execution of following instructions have been completed:

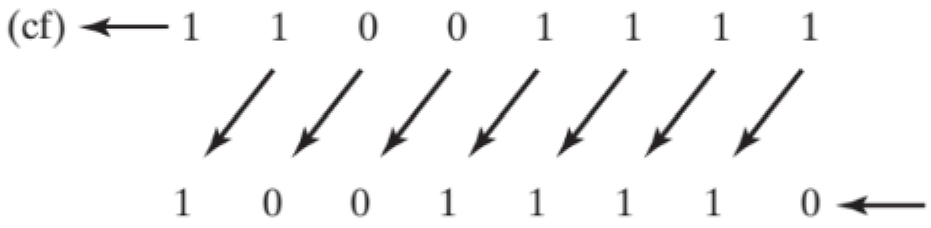
```
SHL     AX, 2
PUSH    AX
ROL     CX, 2
PUSH    CX
RCL     BX, 1
PUSH    BX
XCHG    DH, DL
PUSH    DX
POP     BX      ; BX = CDEF h
POP     CX      ; CX = 579A h
POP     DX      ; DX = D048 h
POP     AX      ; AX = 59E0 h
```

COAL LOGICAL REASONING QS

Shift Left SHL:

•SHL Instruction performs a logical left shift on the destination operand, filling the lowest bit with 0.

•Shifting left n bits multiplies the operand by 2^n.

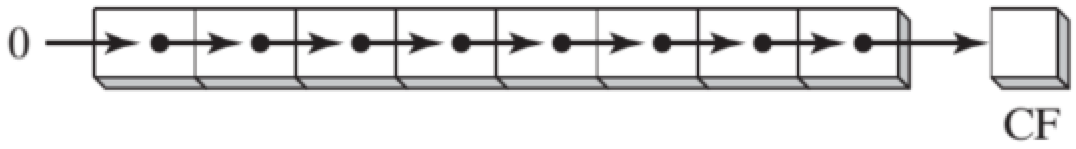


SHL reg, imm8
SHL mem, imm8
SHL reg,CL
SHL mem, CL

Shift Right SHR:

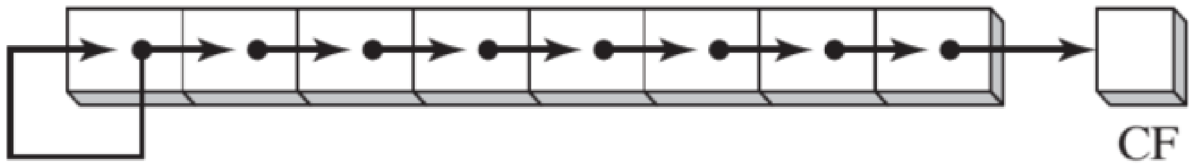
•SHR Instruction performs a logical right shift on the destination operand, replacing the highest bit with a 0.

•Shifting right n bits divides the operand by 2^n.



Shift Arithmetic Right SAR:

•An arithmetic shift preserves the number's sign.



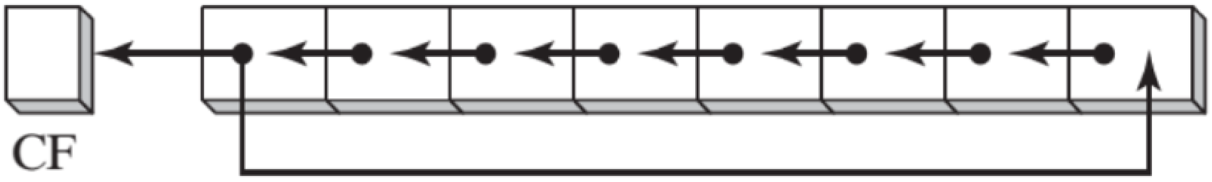
mov al,0F0h ; AL = 11110000b (-16)
sar al,1 ; AL = 11111000b (-8), CF = 0

mov dl,-128 ; DL = 10000000b
sar dl,3 ; DL = 11110000b (-16)

COAL LOGICAL REASONING QS

Rotate Left ROL:

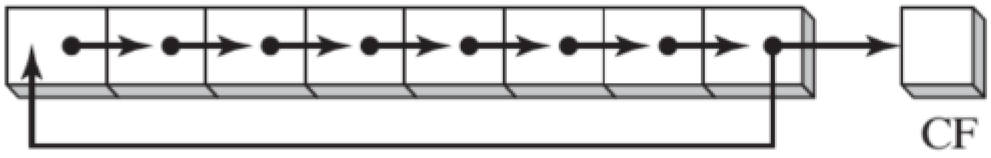
•The ROL (rotate left) instruction shifts each bit to the left. The highest bit is copied into the Carry flag and the lowest bit position.



mov al,40h ; AL = 01000000b
rol al,1 ; AL = 10000000b, CF = 0
rol al,1 ; AL = 00000001b, CF = 1
rol al,1 ; AL = 00000010b, CF = 0

Rotate Right ROR:

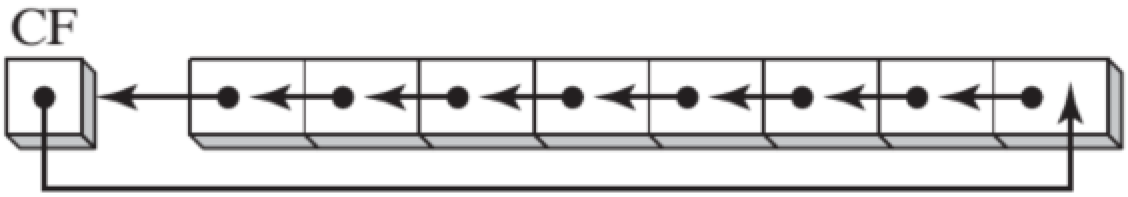
•ROR Instruction shifts each bit to the right and copies the lowest bit into the Carry flag and the highest bit position (MSB).



mov al,01h ; AL = 00000001b
ror al,1 ; AL = 10000000b, CF = 1
ror al,1 ; AL = 01000000b, CF = 0

Rotate Carry Left RCL:

•The RCL (rotate carry left) instruction shifts each bit to the left, copies the Carry flag to the LSB, and copies the MSB into the Carry flag.

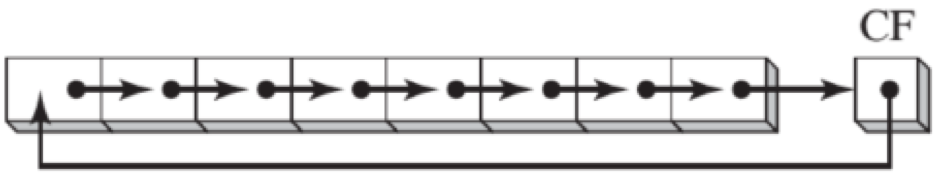


clc ; CF = 0
mov bl,88h ; CF,BL = 0 10001000b
rcl bl,1 ; CF,BL = 1 00010000b
rcl bl,1 ; CF,BL = 0 00100001b

COAL LOGICAL REASONING QS

Rotate Carry Right RCR:

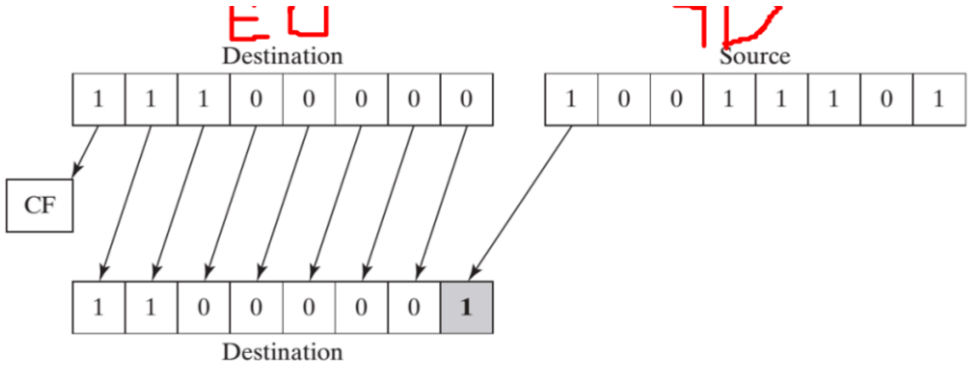
The RCR (rotate carry right) instruction shifts each bit to the right, copies the Carry flag into the MSB, and copies the LSB into the Carry flag:



stc ; CF = 1
mov ah,10h ; AH, CF = 00010000 1
rcr ah,1 ; AH, CF = 10001000 0

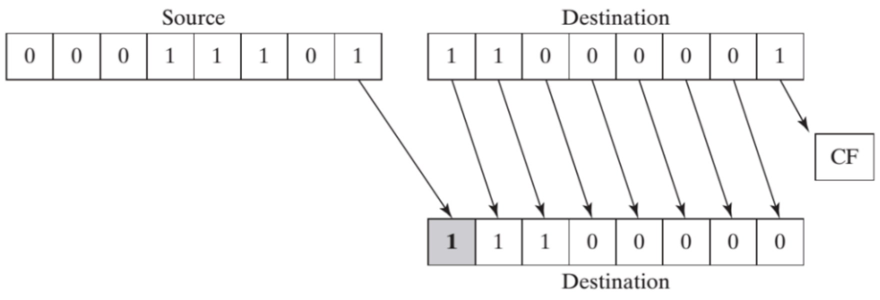
Shift Left Double SHLD:

•The SHLD (shift left double) instruction shifts a destination operand a
given number of bits to the left.
•The bit positions opened up by the shift are filled by the most significant bits of the source operand.
SHLD dest, source, count



Shift Right Double SHRD:

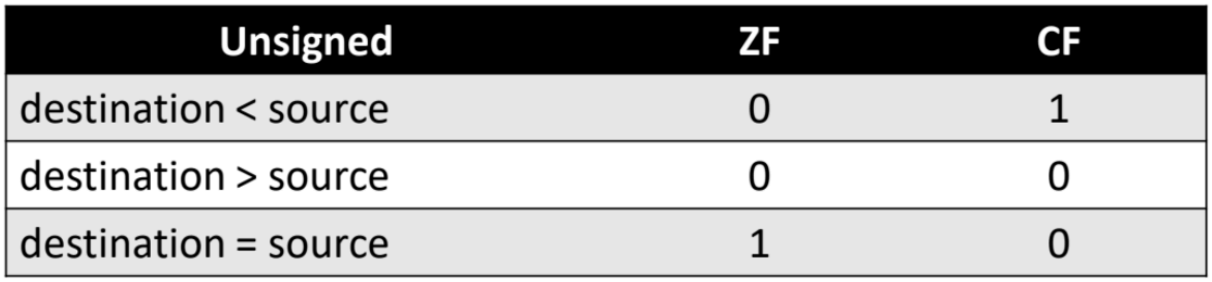
•The SHRD (shift right double) instruction shifts a destination operand a given number of bits to the right.
•The bit positions opened up by the shift are filled by the least significant bits of the source operand:
SHRD dest, source, count



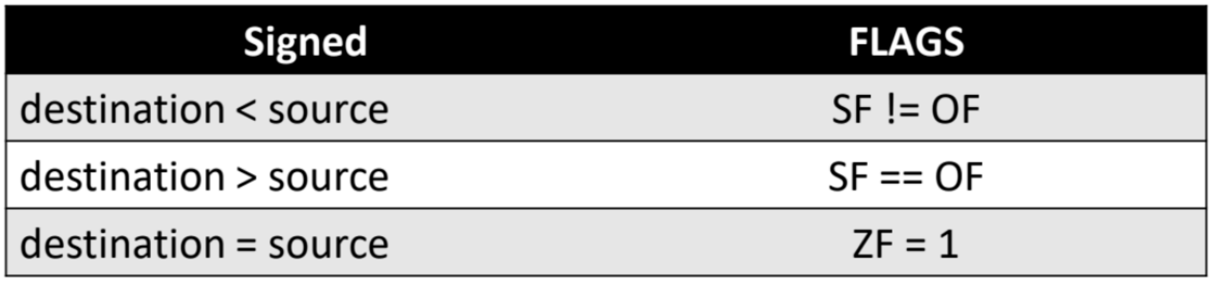
COAL LOGICAL REASONING QS

•The source operand is not affected, but the Sign, Zero, Auxiliary, Parity, and Carry flags are affected.
•The following instruction formats apply to both SHLD and SHRD:

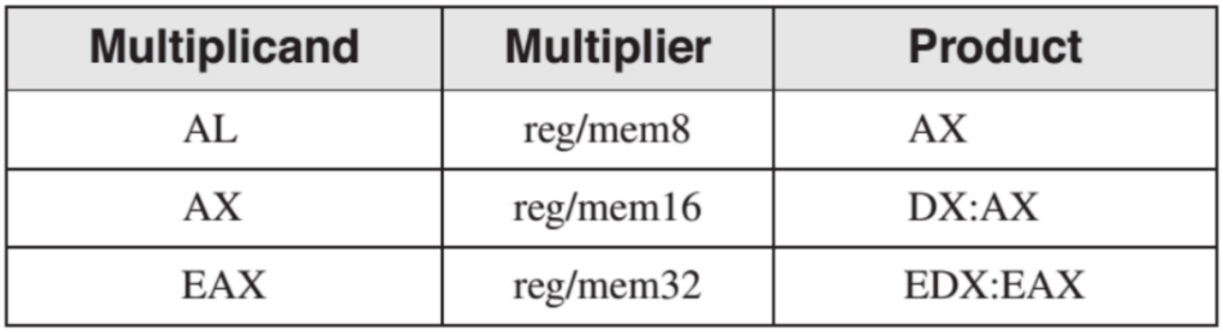
```
SHLD reg16, reg16, CL/imm8
SHLD mem16, reg16, CL/imm8
SHLD reg32, reg32, CL/imm8
SHLD mem32, reg32, CL/imm8
```



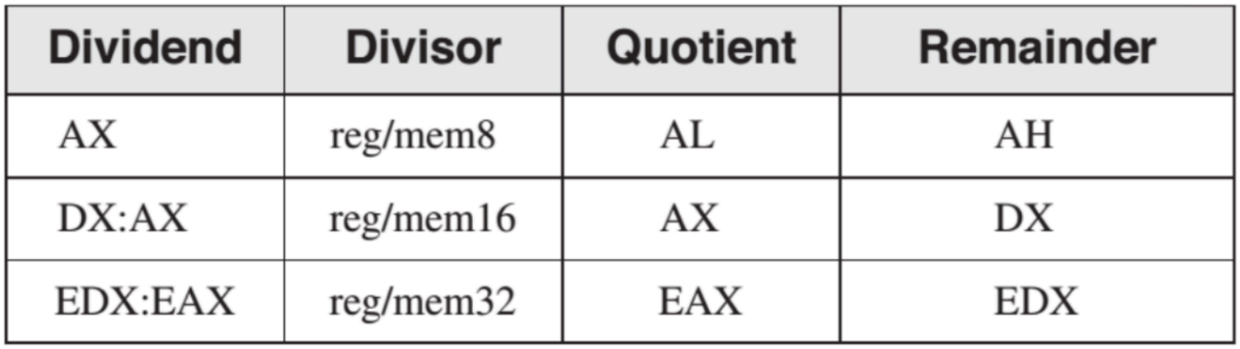
Unsigned	ZF	CF
destination < source	0	1
destination > source	0	0
destination = source	1	0



Signed	FLAGS
destination < source	SF != OF
destination > source	SF == OF
destination = source	ZF = 1

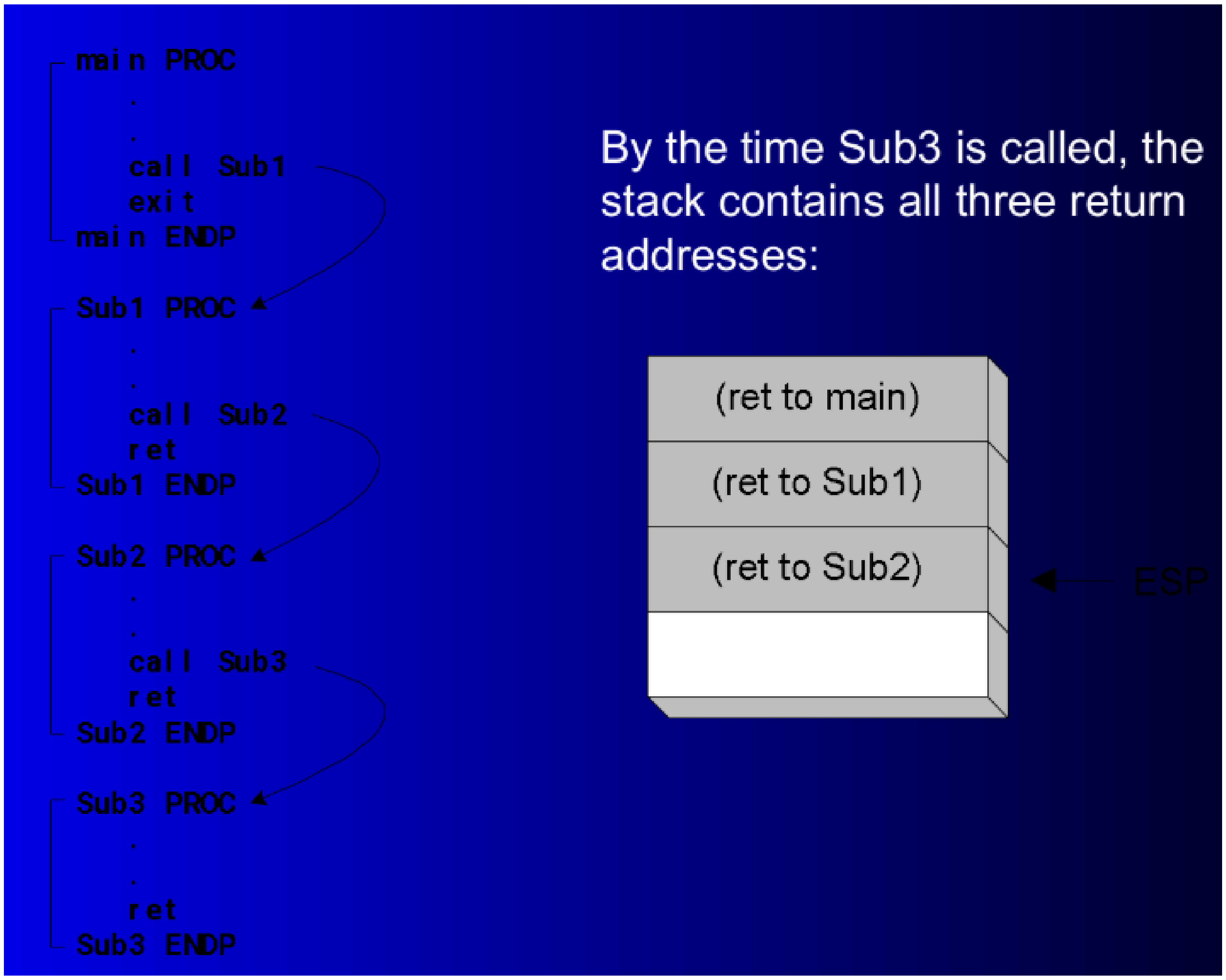


Multiplicand	Multiplier	Product
AL	reg/mem8	AX
AX	reg/mem16	DX:AX
EAX	reg/mem32	EDX:EAX



Dividend	Divisor	Quotient	Remainder
AX	reg/mem8	AL	AH
DX:AX	reg/mem16	AX	DX
EDX:EAX	reg/mem32	EAX	EDX

NESTED PROCEDURES:



```
main PROC
    .
    .
    call Sub1
    exit
main ENDP

Sub1 PROC
    .
    .
    call Sub2
    ret
Sub1 ENDP

Sub2 PROC
    .
    .
    call Sub3
    ret
Sub2 ENDP

Sub3 PROC
    .
    .
    ret
Sub3 ENDP
```

By the time Sub3 is called, the stack contains all three return addresses: