2/5/18

Chapter 1:

Application:

**Application level (7)---> High Order Language(HOL-6)---> Assembly Level(5)---> Operating System(4)---Instruction Set Architecture--->(ISA-3)---> Microcode(2)---->Logic Gate(1)**

Level of Abstraction:

* Suppression of details to show the essence of the matter
* An outline structure
* Division of responsibility through a chain of command
* Subdivision of a system into smaller subsystems

Diagrams(Notebook)

I-input

P- processing

O- output

Software:

**Algorithm- a set of instruction that, when carried out in the proper sequence, solved a problem in a finite amount of time. \*solution**

**Program- an algorithm written for execution on a computer. \* processes**

Function of an Operating System:

* File management
* Memory management
* Processor management

File contains:

* Document
* Program
* Data

Analysis vs. Design :

**Analysis- The input and processing are given. The output is to be determined.**

**Design- The input and desired output are given. The processing is to be determined.**

Byte - 8 Bits

7 Bits

Parity Bits - error checking

**The system performance equation:**

**(time/program)=( instructions/program)\*()cycles/instruction)(\*time/cycle)<---(T=1/f)**

**The bandwidth equation:**

**information=( information/time) \* time**

QR codes- is representation of a 8 bit codes

32 levels of nested programs

Chapter 2:

The C memory model

* Global variables- fixed location in memory
* Local variable and parameters-run-time stack
* Dynamically allocated variable- heap

\*C/C++

Pre Processor

{

-Global Variables

-Libraries = pre-written code that can be accessed by user

#include \_\_\_\_\_

Same location as the source code “STOAFX.h”

If outside <IO STREAM> on Windows

“STDIO.h” on Mac

-Declarations= USING NAMESPACE STD(STANDARD) STD::CIN

= USING NAMESPACE SYSTEM;P

-Prototypes for functions/functions= (user defined)

}  
INT Main()

{

Stack

Data coming back when function ends

Name of function

Parameters data we are receiving from another program

RETURN 0;

}

Assignment

{

Int num;

num = 5;

num= num2+num3

num=num+1

}

Flow of Control

PrintF (“Can print text and number %d, ln”, num);

Scanf (“%d”, &num);

Function Call without void

* Push storage for the return value
* Push the actual parameters
* Push the return address
* Push storage for the local variable

Function Return

* Pop local variables
* Pop the return address and use it to determine the next instruction to execute
* Pop the parameters
* Pop the return value and use it as specified the next calling statement

Function Call with void

* Push the actual parameters
* Push the return address
* Push storage for the local variable

Function Return with void

* Pop the return address and use it to determine the next instruction to execute
* Pop the parameters
* Pop the return value and use it as specified the next calling statement

Computer processes by sequential

Relational operators

> less than 5 > 4 t

< greater than 5<4 f

> = less than or equal to 5 >= 4 t

< = greater than or equal to 5 < = 4 f

== equal to 5==4

!= not equal to

Logical operators are used to combine relational statements

And &&

Or || (pipe symbol)

Not !

(5>4)&&( <20)

Conditional

Decision-- if and if/else

Selection of menu of choices-- switch and elseif

Loops --Repetition

Counting loops

For ( x=0; x<5;x++)

{

Whatever  
}

For x=0 to 5

Next x

Pre-test

While- condition must be true to process

Post-test

Will process once and then check the condition

Call by reference

In call by value, the formal parameter get the value of the actual parameter

If the formal parameter changes, the actual parameter does not change

In call by reference, the formal parameter gets a reference to the actual parameter.

If the formal parameter changes, the actual parameter does change.

Pointers:

&= outputs

\*= reference for an address

%= inputs

/n = new line

Microsoft for Visual Studio:

#include “stoafx.h”

#include <iostream>

using namespace System;

Using namespace std;

Two operators for dynamic memory allocation (102)

malloc() = to allocate from heap

free()= to deallocate from heap

Ch3

Decimal10

Octal8

Hexidecimal 16

Binary 2

Status Bits - always show in this order

N=0 otherwise (Negative number N=1; Positive=0)

Z= 1 is all 0’s

V= if signed integer overflow occurred

C= if unsigned integer overflow occurred

Operators

Arithmetic

Neg

Relational

Or

XOR

And

TT=T

TF=F

FT=F

FF=F

Both true to be true

Or

TT=T

TF=T

FT=T

FF=F

Only 1 need to be true

XOR- Exclusive OR

TT=F

TF=T

FT=T

FF=F

One and only one of any

CH4

Purple tab Pg 185

Word size- is the amount of word that a computer can process at one use

Big Endian =Lowest & Little Endian = Highest

The order of the sequence of bytes in main memory.

Sign #’s - can have overflow

unsign #’s- no overflow

Both can have carry bits

1 Adding negative and positive→ never overflow v is always 0

2 sign/carry same→ V always 1

sign/carry different→ V always 0.

FC15= input

FC16= output

Data→ CPu→ memory

In ← CPU← memory

Commands:

Stop

00

0000 0000

\*\*\*\*Some are on the book highlighted and some are on notes\*\*\*\*

The von Neumann execution cycle

1. Fetch instruction at Mem[PC]

2. Decode instruction fetched

3. Increment PC

4. Execute the instruction fetched

5. Repeat

Non-unary Instructions

Instruction= 1 Byte Operand = 2 Bytes

Reading Data

C1 - Loading a word

Memory location where data is located (will always read 2 bytes)

Or FC15- input device

D1 - Load byte

Memory location will only read 1 byte (1 hex number or 8 bits)

Or FC15- input device

Store Data

E1

F1 FC16

Writing a program

Input first Number

--- Add 2nd number to the accumulator register

---Add 3rd number to the accumulator register

---Convert the decimal number to ASCII -Using a mask

Tell compiler to apply the mask (give location of the mask)

--- Print the answer- Store the contents of the accumulator register (masked) to the output device

1 thing at a time

Fetch - Load- stores the data in a register- accumulator

- overwrite it - store to output device? First

add to it

Accumulator Register- can accumulate things

Writing a program

0000 C1 00 0D- First number

0003 61 00 0F- 2 number

0006 91 00 11- mask

0009 F1 FC 16- print it

000C 00

000D 0002

000F 0002

0011 0030

zz

0000 C1 00 10

0003 61 00 12

0006 61 00 14

0009 91 00 16

000C F1 FC 16

000F 00

0010 0002

0012 FF FD

0014 00 06

00 16 00 30

zz

Mask is always 30

Trap Handler= it passes the instruction to the operating system.

Assembly Language Ch. 5

Mnemonic Letter Data

Instruction Addressing Pseudo oPs(Dot Commands)

61 0009= 0009 Data; ADDA= Add to the accumulator register 0 x 0009= The data located here; d = use direct addressing

; = this a comment or annotation

Pseudo- Op. Hexadecimal

LDBA 0x000D, d ; D1 0009

STBA 0xFC16,d ; F1FC16

LDBA 0x000E,d ;D1 000E

STBA 0xFC16,d ; F1FC16

STOP ;00

.ASCII “HI” ;4869

.END

.BLOCK= A block of zero bytes.

Num1: .BLOCK 2 <= 2 digits number

= int num 1;

DECI num1, d = decimal input

Mnemonic operand-same as in Hex(usually), addressing mode- 8 possibilities

LDBA 0X FC15,d

1101 0001

Instruction do what?

What register?

What type of addressing?

.BURN\*-initiate a rom burn

.END\*- sentinel for end of assembler

.EQUATE\*- equate a symbol or constant values

.ADDRESS- address of a symbol

.BLOCK- block of 0 bytes

.BYTE- byte value

.AIIGN- padding to AIIGN boundary

.ASCII-”Hello”

.WORD-Word value

**ADDRESS OBJECT CODE SYMBOL MNEMONIC OPERAND ; COMMENT**

ASL(A)= multiply

ASR(A)= divide

operand= \_\_\_\_\_ of address of the memory location

LDBA ‘H’, (d,i)

000 00 48

Instruction

d(direct addressing)- is memory location and put it on the bus tro be loaded to accumulator

i( immediate)- is the operan stores in main memory without doing a memory fetch

Trap instructions

(more than one digit)

Deci = decimal input

Deco= decimal output

Br= Branch

Instructions generates in bits

Hexo= Hexidecimal Output

Hexi= Hexidecimal Input

PrintF(“Hello World! \n”);

Return 0;

Assembly:

STRO msg,d

STOP

msg .ASCII “Hello World!\n\x00”

.END

-----------------------------------------------------------------------------------------

CHAR CH;

INT NUM;

SCANF(“%c %d”, & CH, &NUM);

CH .BLOCK1

NUM .BLOCK2

;

LDBA 0XFC15,d; LDBA CHARIN,d

STBA 0X0003,d; STBA CH,d

DECI 0X0004,d; DECI NUM, d

------------------------------------------------------------------------------------------

CH++;

CH+=3

LDBA CH,d

ADDA 1,i

STBA CH,d

CH=CH+1

------------------------------------------------------------------------------

PRINTF(%c\N%d\N”, CH, num);

LDBA CH,d

STBA CHAROUT,d

LDBA ‘\n’,i

STBA CHAROUT,d

DECO NUM,d

LDBA ‘\n’,i

STBA CHAROUT,d

---------------------------------------------------------------------------------------

Symbol Table

Name Address Type of Data

CH sChar

Num 0x0048 sInt

Main 0x0005

num=num%2;

main()- stack addressing allows function to access the information that was pushed onto the stack.

Operand←-- mem(Sp+operand specifier)

Pep/9 stack starts at FB8F

ADDSP- Add to stack pointer

SUBSP- subtract to stack pointer

CALL- call

RET- return from call

RETTR-return from trap

LDWA num,s

BRLT place

BRLT- test the n bits if it is 1 it will branch to place

Branch is the if-condition.

CPWA-compare word accumulator

\*Subtract the operand to accumulator

Num1 .Block 2 - reserves a byte of storage (global

Deci- just from the keyboard

Num2 .equate 155 -> constant

Num3 .equate

Declaring variables

* Setting aside storage for variables
* Getting the data into the memory, onto the stack

.Block - global

Subsp- pushes local variable onto the run-time stack

Sp= stack pointer

subsp\_\_\_\_ , i

Function call use “call”